Osculant:
A Multiprocessor Self-Organizing Task Scheduler

Final

Project Report

Contract Number: N00039-94-C-0163

Dr. Fred J. Taylor

High-Speed Digital Architecture Laboratory
Department of Electrical and Computer Engineering
University of Florida

February 17, 1998

Contents

Osculant[1]: A Multiprocessor Self-Organizing Task Scheduler

Project Report

High-Speed Digital Architecture Laboratory
Department of Electrical and Computer Engineering
University of Florida
Gainesville, FL 32611
February 17, 1998

1 Introduction

Heterogeneous network computing is now ubiquitous and is found in virtually every major computing environment. Heterogeneous computing is known to be cost-effective, robust, and scaleable. With Moore Law [Schaller 1997] driving technological improvements, the entire field of heterogeneous computing is undergoing a metamorphosis. Nevertheless, within this dynamically changing landscape, there is a need to provide management of these computational resources. Computer engineers unknowingly have, for some time, been introducing market driven philosophies into their design strategies. These include organization theory (e.g., shared resources), recycling (e.g., cache), commodity investment (e.g., speculative computing), to name but a few. Current studies have indicated that market driven concepts can, in fact, be integrated formally into a resource management paradigm for heterogeneous computing systems. What is important to realize, is that network computing suffers from the same set of restrictions that govern supply-side economics in terms of access to resources and services and hierarchical control. These system-level attributes and resources, which are

[1] Osculant = Intermediate in character between two related or similar taxonomic groups, closely joined.

important to the conducted study, relate to:

- Bandwidth and latency: It is self-evident that network bandwidth restrictions can seriously impair timely execution tasks. Another performance limiting observation is that network latency (i.e., the time required to receive requested information) is not directly correlated to communication bandwidth when the message lengths are small.
- Localized information and service providers: In order to avoid unnecessary network traffic congestion, information storage should be distributed, or duplicated, in a prudent manner. In this way, a balance is achieved between system performance and system cost.

These observations point out that task and resource scheduling will play an important role in improving the performance of virtually any network-based computing environment. A new bottom-up resource scheduling paradigm, call Osculant, was proposed as an innovative facilitating technology. Osculant was developed under contract NAVY 00039-44-C-0163 and is reported under this cover.

Many studies can be related to the developed Osculant paradigm. Load balancing and resource allocation are key topics in designing efficient multiple-node system. Classic load balancing algorithms [Goscinski 1991] as well as other novel approaches, such as the microeconomic load balancing algorithm by Ferguson [Ferguson 1988], focus on closely-coupled or homogeneous computing systems and generally concentrate on the task processing time. Shin et al. [Shin 1988 & 1995] proposed a resource allocation policy which uses buddy sets to reduce the state-collection overhead in a multicomputer configuration similar to the target environment. In Osculant, conversely, we exploit the state-probing approach with self-regulating task announcement processes [Smith 1980] and aggressive bidding [Ramamritham 1989], [Smith 1980], [Ni 1985] strategies. Ramamritham [Ramamritham 1994], Blake [Blake 1991], and Liu [Liu 1973] studies motivate the development of bidding strategies in the Osculant. These task scheduling schemes, however, are restricted to hard-real time environments where most of task and resource requirements are well known prior to the scheduling time. Osculant, on the other hand, relates to a general-purpose distributed computing where execution time constraints are not critical and new tasks are allowed to enter the system. Other related topics include job profile extraction techniques [Puschiner 1989], [Park 1991 & 1993], [Shaw 1989] and task announcement distribution designs [Chow 1996], [Goscinski 1991], [Shin 1995].

The final report is organized in the following manner: Section 2 describes the basis of the Osculant scheduler. Section 3 presents comparisons between top-down and bottom-up scheduling schemes. In Section 4, a review of job profile generation techniques are presented. Section 5 discusses the Osculant jobpost distribution protocols. In Section 6, various bidding strategies are explained and some results are presented. Section 7 discusses the resource management schemes in an Osculant system. Section 8 and 9 describe the developments and designs of Osculant Simulator and Osculant Shell. Section 10 discusses the future works, research topics of the Osculant scheme. Finally, in Section 11, the conclusions are presented.

2 Overview of The Osculant Scheduler

Osculant differs from existing task and resource scheduling paradigms in that it is bottom-up and self-organizing. Experimental studies have led to the conclusion that Osculant is: (1) architecturally robust, (2) capable of internalizing the management of system assets and, (3) able to dynamically alter the system ethos to range from a real-time operation, to maximize bandwidth, to minimize latency, to minimize energy dissipation. The Osculant paradigm can be motivated as follows:

- An Osculant system consists of a collection of possibly dissimilar autonomous information systems (e.g., capabilities, instruction set, local storage, I/O) which may or may not be connected by a network with an arbitrary topology and time- and space-varying behavior.
- Executed programs send messages to a higher level entity, called the *steward,* which interprets these requests in terms of executable objects and posts them on a *job board* along with salient information about data location, resource requirements, job priority, and so forth.
- When a job is posted, all processors bid on that job in a manner which maximizes their profit (measured in terms of tangibles such as net number of cycles per unit time). Job bids are a function of processor resources, locality of data, I/O costs, job priority, existing local job queue, and so forth. Processors have no knowledge of other bids and operate autonomously. The *steward* receives bids and awards tasks to the processor with the best bid.
- Any processor can play one of the three following roles:

1. User: A user node issues jobs.
2. Steward: A steward node manages jobs authorized by other users or assigned by stewards.
3. Participant: A participant node bids jobs and executes the job once assigned by the steward.

- Role assignments of nodes are based on individual jobs and circumstances . Hence, a node can play one or more roles at the same time for different jobs.

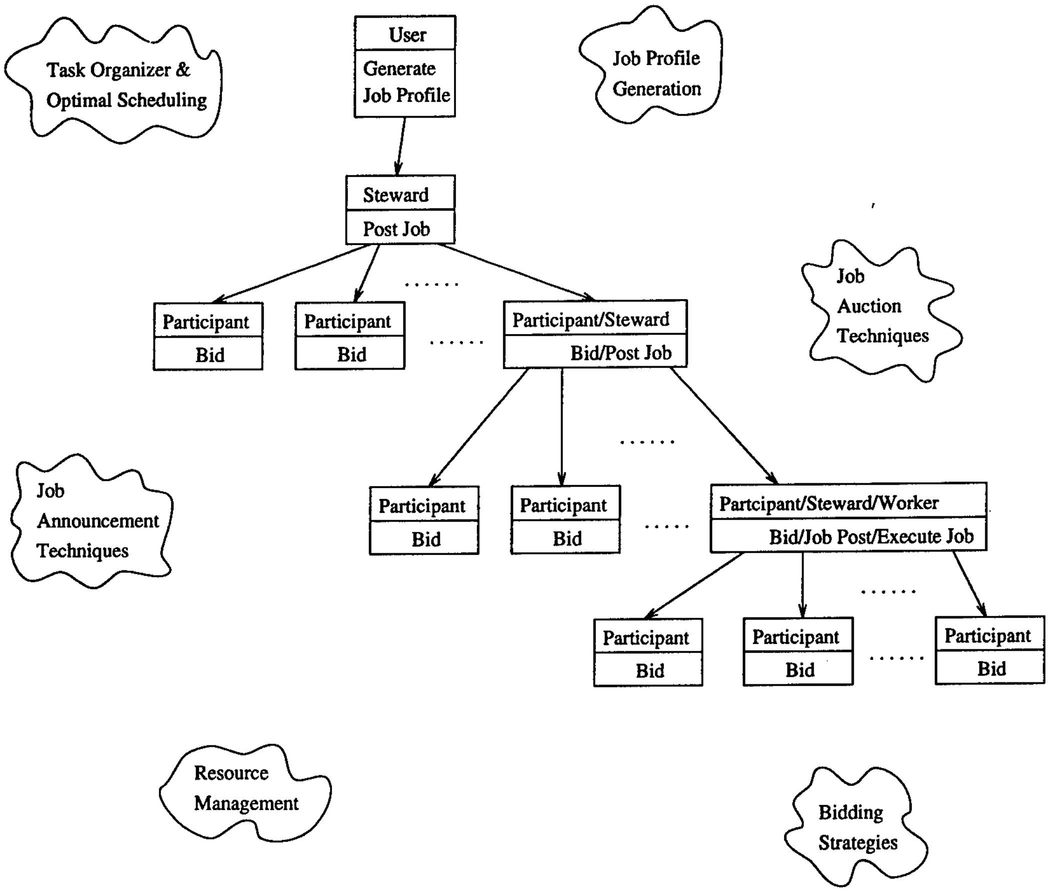

Figure 2.1 Overview of the Osculant scheduling scheme.

Figure 2.1 shows an example of the basic operations of the Osculant scheduler and the related study areas. It should be noted that Figure 2.1 only shows the logical hierarchical structure of the Osculant system. In reality, the connection scheme can be graph, ring, tree, or a combination of them.

The basic market-driven philosophy underlying an Osculant system is a competitive bidding scheduling scheme. The key to successful bidding is to estimate accurately and efficiently the cost-complexity of a posted job. This is the role of the job profile generator (JPG). Job profiles are first generated from the information provided by the tasks which are distributed among participating nodes. Upon receiving the job profiles, the distributed heterogeneous nodes calculate the completion cost of a task based on their interests, ethos, biases, and capabilities. A job will be assigned to the node by the *steward* (which is re-locatable) with best bid.

3 Comparisons Between Top-down and Bottom-up Schemes

The Osculant is a bottom-up scheme that is capable of assimilating the most up-to-date system information and, therefore, achieves optimal scheduling performance. Performance optimization is attained with the possible overhead expense associated with a relatively long time period spent in jobpost/bidding process. In this section, the characteristics of jobpost/bidding delay on the system performance are studied.

In order to show the effects of jobpost/bidding delays, three top-down schemes are compared to the Osculant bidding scheme:

- Round robin: Jobs are assigned to working nodes in a sequential order among working nodes.
- Random: Jobs are distributed randomly among the working nodes.
- Well informed top-down scheduler: The scheduler has complete knowledge about the capabilities of all working nodes but does not have information about the load generated locally at the nodes. This scheme also serves as the an ideal comparison counterpart to the bidding scheme because it accurately estimates working node information without any jobpost/bidding delay if there is a lack of local activity at every processor.

The results shown in this report are retrieved from a sequence of simulations. In the simulator, a collection of jobs are generated and fed into the scheduler with various local loads (jobs that are generated by local users which are unpredictable to the top-down scheduler) and system parameters (which are known to the schedulers). Jobs are characterized by job size, and by similarities between jobs and job generation rate.

The first simulation shown in Figure 3.1 assumes a homogeneous system with four identical processors. Local job are added gradually to one of the working processors while they keep the local load of other nodes constant. The results show that the performance gain of random scheduler remains random; the round robin scheduler and the well informed top-down scheduler have approximately the same performance; the Osculant scheduler gradually performs better as the level of unbalanced load increased. These results show that the Osculant scheme can adapt effectively to the current processor status under this condition. The effect of post/bidding overhead also can be observed from this figure: When the homogeneous system is well balanced and lightly loaded, both the round robin scheduler and the well informed top-down scheduler out-perform the bidding scheme.

The same simulation is also performed in a heterogeneous system of four processors with different service rate (1:2:3:4 in this example). Simulation is achieved by manipulating the local job load as in the homogeneous case. As shown by the results, the advantage of the Osculant scheme becomes more prominent. In this case, both the random and round-robin schedulers have worse performance while the Osculant scheduler generally outperforms the well-informed top-down scheduler by around 20%. This is mainly because the bidding scheme can retrieve more accurate local information than the top-down schemes. When the well-known top-down scheduler distributes a job to a heavily loaded processor, the performance penalty to the system is more severe in a heterogeneous system than in a homogeneous system.

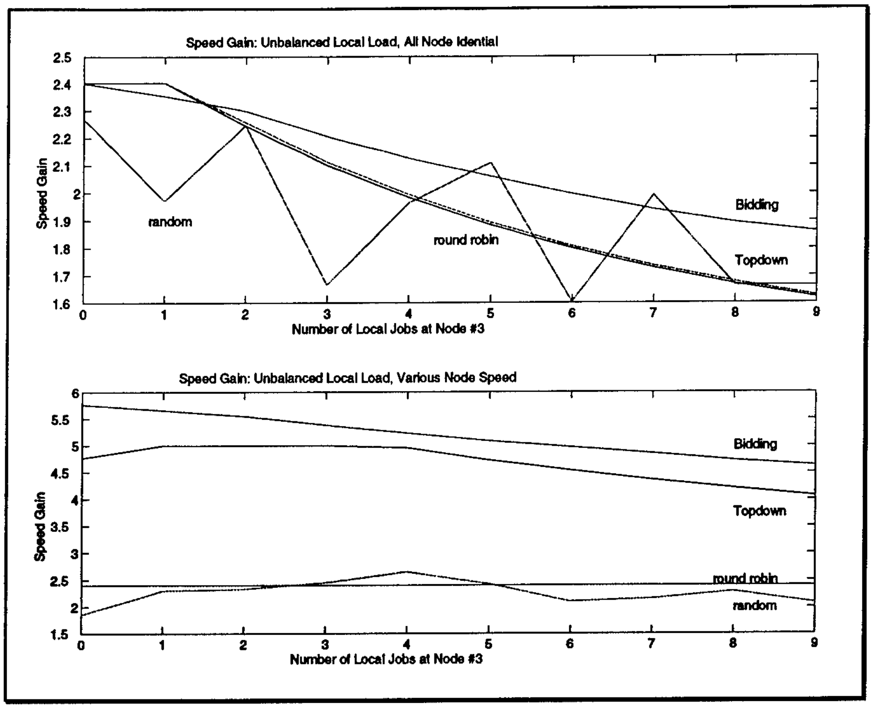

Figure 3.1 Comparisons between top-down schedulers and the bidding scheme. In these results, the bidding scheme outperforms the top-down schemes when the system load becomes unbalanced. The bidding scheme has more prominent performance advantages when the system contains processors with diverse service rate.

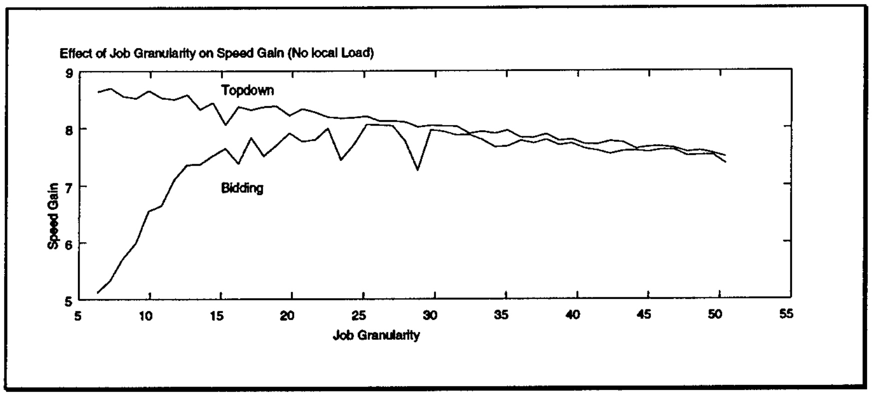

Figure 3.2 The effect of job granularity on the performance of bidding scheme. The performance of the top-down scheduler is an ideal case counterpart if the jobpost/bidding delay is not present.

The simulation reported in Figure 3.2 investigates how the jobpost/bidding overheads affect the overall system performance. In this simulation, the well informed top-down scheduler has accurate knowledge of the working node status and lacks a local job load in the working nodes. Under this condition, the quality of job distribution by the host processor is comparable to the bidding scheme but lacks bidding overhead. Therefore, the well-informed top-down scheduler serves as an ideal case for the Osculant scheduler. The other parameter under control is the job granularity. Here, a job is partitioned into different sizes and the overall completion time is measured. From the results, as job granularity increases, performance of bidding scheme gradually approaches the ideal top-down scheduler. Conversely, large job granularity will result in low performance gain because of low degree of parallelism.

These studies suggest that Osculant can perform as well as the best top-down scheduler, as well as offering its unique properties and attributes.

4 Osculant Job Profile Generator

The basic market-driven philosophy underlying Osculant is one of competitive bidding. The key to successful bidding is to estimate accurately and efficiently the cost-complexity of a posted job. This is the role of the Osculant Job Profile Generator (JPG). Job profiles must be generated first from the information provided by the tasks which are distributed among participating nodes. Upon receiving the job profiles, distributed heterogeneous nodes calculate the completion cost of a task based on their interests, ethos, biases, and capabilities. The job will be assigned to the node by the *steward* with the best bid. The quality of JPG directly affects the bid accuracy and scheduling performance. The Osculant JPG differs from known real-time task profile generators [Shaw 1989], [Park 1993], [Shin 1988] in a number of important ways. The JPG has been designed to be platform independent for use in a dynamically changing and/or fault vulnerable real-time environment.

Meeting deadlines is critical in real-time applications. A desired condition is to assimilate new tasks into the computation environment without altering the systems ethos. Conceptually, it is important to develop a methodology for producing both reliable and deterministic task timing and resource requirement predictions directly from high-level application code. From the generated profile, the distributed computational nodes are able to estimate the task completion cost based on the JPG and knowledge of their local state.

4.1 Design Principles and Algorithms

A JPG is responsible for creating job profiles that act as a unidirectional bridge between tasks and announce their presence to the distributed heterogeneous computing resources found within a networked system. The basic elements in the task domain include:

- Priori execution job profiles: The job profiles of announced tasks will be made available to a heterogeneous system prior to the job execution phase. In the case of Osculant, profiles are created prior to the bidding phase. Numerous studies and commercial products generate program profiles during or after the execution phase. The reported JPG, in contrast, generates profiles prior to the job execution phase.
- Inputs space: It is assumed that sufficient task information resides at the source code level in order to quantify the complexity and the data needs of a job. The profile generator itself may need to integrate other profiles into the JPG production process. Bidding process will also utilize the JPG data as well as the information concerning data locality and network health to develop a bid.
- Architectural specifics: In a heterogeneous computing environment, nodes can have different configurations, memory capacities, I/O capabilities, processing unit combinations, and designated roles to play within a system. The job profiles should carry relevant architectural information about job specification and execution constraints. The profiler must allow diversified nodes to calculate the job completion cost and to accommodate node capabilities and ethos.

From this information space, a bidding processor, or node, must interpret the job profile of a task that requests service to create a bid. The bid, in turn, is defined in terms of the estimated resource requirements, state of scalar control variables, data organization, plus other attributes supplied by the JPG. Regardless of the details, the profile database must be kept as small as possible. This rationale is based on the fact the job profiles will be transmitted to bidding nodes distributed within a network. Secondly, the JPG database should not be highly invasive at the compute-server side. Otherwise, the resources required to bid and/or manage multiple jobs would become too expensive.

4.1.1 Function Result Type Tables

Function Result Type Tables (FRTT) store information regarding system functions. In some cases, entries in FRTT are functions that are previously profiled. Functions listed in the FRTT will be processed faster than from their source code once they are properly characterized. The FRTT can be modified during run-time and improve the estimation accuracy and execution efficiency. FRTT contains two libraries what are (1) the Function Result Size Table (FRST) and (2) Osculant Function Profile (OFP). The format of FRST is:

Function Name	Rule	Confidence	Number of Inputs	Group

The entries are as follows:

- Function Name: Label of a function or service.
- Rule: Methods applied to determine the size of output variables. These rules defined include SAME, MAX, GEN and IO.
- Confidence: A number that represents the confidence in the estimate of output variable size determined by the rule and input variables. A confidence of 1 means the

output size is determined completely by the rule and the inputs. A confidence of 0 means the output is scalar. The initial confidence values in FRST are chosen heuristically.

- Group: Group introduces cross-referencing capabilities to the profile generator. This is required since some functions may behave differently when coupled with other functions. In some cases, the rule and confidence of some functions can be determined by a previously profiled function. This cross-reference capability simplifies profile modification and enhances search efficiency.

For example, in a MATLAB environment, FRST record of the expectation function mean() is "mean SAME 0.1 1" which is interpreted to mean that: "Function mean needs one input variable. Outputs of this function are of the same size as the input variable with confidence of 0.1." The expectation function in MATLAB can take 1-D or 2-D matrix as inputs. Therefore, the results can also be a scalar or a 1-D array. Appendix B shows examples of FRST for MATLAB version 4.0.

While FRST concentrates on the variable size estimation, the Osculant JPG focuses on estimating the function execution time. The OFP for MATLAB, for example, estimates the number of floating point operations (flops) using polynomial curve fitting method [MATLAB 1992]. Polynomials coefficients are computed using pairs of inputs (size of input variables) and outputs (flops) so that the mean-square errors are minimized. The format of the Osculant Function Profile (OFP) is as follows:

Function Name	Polynomial Coefficients	Previous Execution Results

Some execution times from previously executed functions are stored in the OFP. Therefore, the polynomial coefficients of a function can be constantly calibrated to improve accuracy. The current implementation of OFP uses a polynomial of degree 4 and stores 20 previous flops parameters. The present design of OFP assumes a linear model in estimating computation load. This model is simple, but it does not hold for certain functions. Further discussions will be presented in Section V. Appendix C provides an example of the OFP for MATLAB version 4.0.

The design of FRTT is based on a fuzzy estimation scheme where the result can be determined from several serial (dependent) or/and parallel (independent) estimates. During a variable back tracing process, it is possible to have several ancestor variables directly or indirectly referenced by the target variable. If an estimate has a confidence value of 1, then the value or size of the target variable will be accepted on a *prima facia* basis. Otherwise, the target variable will be estimated by the Location Information Adjustment method (explained in Section 4.1.3). An example provided in Appendix D.

4.1.2 Granularity

It is found that the computation load for many classes of functions are not linear with the size of input variables space. The estimation of the computation load at the bidding nodes will, in these cases, need to be synthesized with some care. A possible solution is to implement a multi-mode, or high resolution Osculant Function Profiler (OFP), that is based on statistics of input variables for all function calls. It is impractical to send detail variable size information in job profiles since this would consume too much network bandwidth. For the JPG, the simplest measure of granularity is used where the granularity function is represented by the number (granularity index) of individual calls. The results obtained to date (shown in Section 6) are promising and take small estimation errors. The problem of nonlinear computation load will be further discussed in Section 4.2.

4.1.3 Supporting Mechanisms

Several supporting mechanisms have been implemented to improve estimation accuracy and efficiency. Most of these processes are application or language independent.

- Location Information Adjustment (LIA): In the assumed programming style, values and size of variables are related to other variables which have previously appeared or been referenced. It is possible, however, that there are multiple references for the target variable. The LIA process uses normalized weighted sum according to the proximity to the target variable to determine the result. An interesting development of the LIA method is lifting the backward reference requirement. It is found that, by referencing the variables appearing after the target variable, the scope of the tracing process is increased. However, the limited improvement in estimation accuracy does not justify the longer tracing time (tracing time of a variable is constant regardless of its location in the program).
- Variable Dimension Adjustment: In many cases, only a portion of a data matrix is used in an operation. The adjustment process utilizes the information extracted by the filtering process and modifies (reduces, in most cases) the variable size. For example, parentheses are good indicators for matrix size reductions. In some cases, a sequence

of program calls may provide some information that can be used to correct estimation error. In another example, by matching input and output parameters between the parent and child processes, errors can be reduced in estimating the contents of parent process.

- Inline Scalar Evaluation: The process will evaluate an instruction line that contains only scalars. It is observed that most scalars are either used to carry important size information or to act as control variables. Evaluating the scalar instruction line will improve estimation accuracy. This process is done by cross-referencing the original program expressions and the estimated results from the Variable Back Trace process. Only scalar results will be returned.
- Cache Design: A simple cache system is implemented to improve variable back tracing efficiency. Experiences show that a variable will be repeatedly referenced or traced in a single variable back trace process. Storing previous execution results will greatly reduce the running time. The same mechanism is also implemented in the Profile Generator where the estimated profiles of subroutines are stored.

4.1.4 Algorithm Issues

For modern structured programming languages, correctly tracing nested loops and branches are axiomatic. The SAM_Generator pre-processor encodes level of nests before executing the main profile generator. With simple stack operations, the previous function statistics can be retrieved and stored correctly.

The heart of the profile generator is the Variable Back Tracing (VBT) engine (see

Appendix E). The VBT recursively traces the variables referenced by the target variable. The stop condition of recursions mostly are input parameters and explicitly defined variables in the program. The FRST, Location Information Adjustment, and Inline Scalar Evaluation method work cooperatively to find the final estimate of the target variable.

The next step is to categorize the functions such that they can be traced properly. The first stage of classification determines the types of the called functions based the FRTT. Functions that are not in the FRTT will be considered as customer functions. The second stage of classification names types of operators and traces input parameters. Memory occupancies of tasks are mostly determined at this stage.

The main JPG program (see Appendix F) maintains and accesses the cache storage. If a cache miss is encountered, the function will call the nest handling process and function classification processes. The profiles of subroutines and functions can be combined to form the final result. The conclusion of loops and branches need more attention. First, for the worst case design principle, the block with the highest computation load in a branch is always taken. Second, iteration numbers of un-deterministic loops are found by referencing the size of the control variables. The overall computation load of a loop is the product of the estimated iteration number and the local computation load. Finding memory occupancy, compared to the calculation of computation load, is much easier. The memory requirements for a job is the maximum amount of memory occupancy during tracing.

4.1.5 Implementation

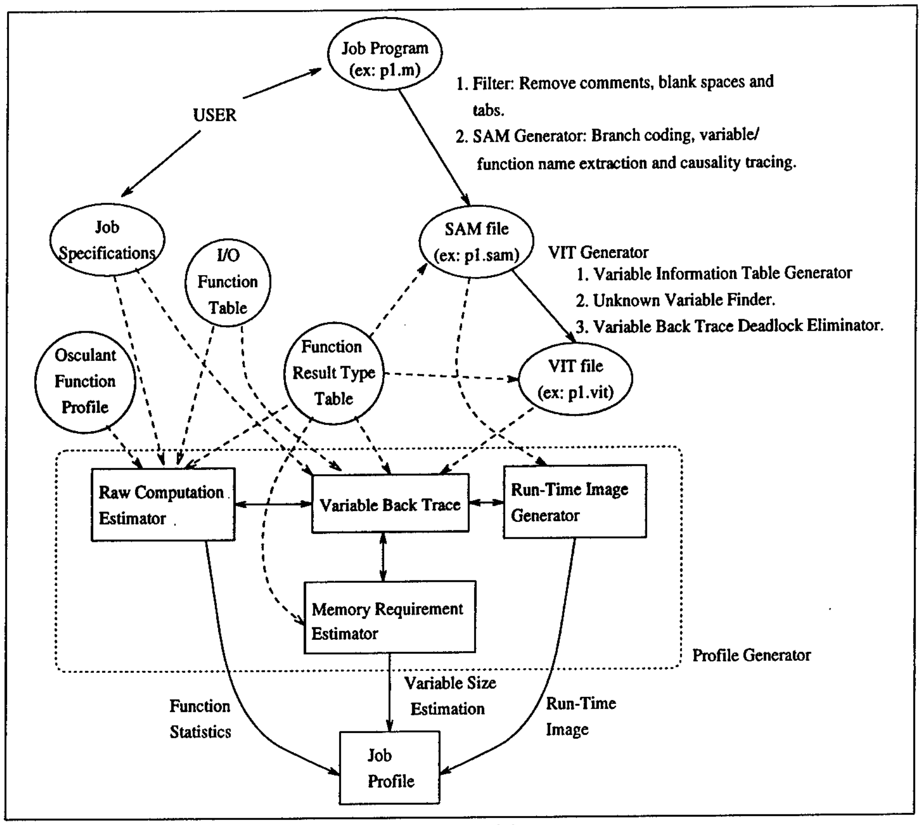
Job Program (ex: p1.m)
USER
1. Filter: Remove comments, blank spaces and tabs.
2. SAM Generator: Branch coding, variable/ function name extraction and causality tracing.
Job Specifications
I/O Function Table
SAM file (ex: p1.sam)
VIT Generator
1. Variable Information Table Generator
2. Unknown Variable Finder.
3. Variable Back Trace Deadlock Eliminator.
Osculant Function Profile
Function Result Type Table
VIT file (ex: p1.vit)
Raw Computation Estimator
Variable Back Trace
Run-Time Image Generator
Memory Requirement Estimator
Profile Generator
Function Statistics
Variable Size Estimation
Run-Time Image
Job Profile

Figure 4.1 Osculant Job Profile Generator structure.

Figure 4.1 shows the structure of the JPG. The individual functions of each block are:

- Filter: This process separates operands and operators. It also eliminates undesired information such as strings.
- SAM Generator: This process performs branch coding, variable extraction, and function name extraction.
- VIT Generator: This process generates the Variable Information Table (VIT) which is a cross-reference table between the variables and functions. An important role of this process is to remove variable back tracing deadlocks. These deadlocks exist in many situations, such as in recursive function calls.

- Job Profile Generator: The Variable Back Tracing engine estimates the value of and the size of variables. In order to determine the value and the size of a variable, it may need to search recursively among one or several Variable Information Tables. The main process follows the program structure and generates job profiles. The JPG is a one-pass process and operates several stacks and lists in order to collect program information.

  JPG outputs contain

  (1) Estimated computation requirements in flops,

  (2) Function statistics with granularity information,

  (3) Run-time image,

  (4) Memory requirement, and

  (5) Profile generator execution statistics.

Figure 4.2 shows an example of the job profile for a test sample. Most of the job profiles are in integer format, are sparse, and can be compressed and efficiently transmitted. The resulting job profile, therefore, not only provides essential information of tasks, but is easily computable, and can be readily implemented in hardware. The current version of the Osculant Profile Generator is implemented in C language and MATLAB 4.0.

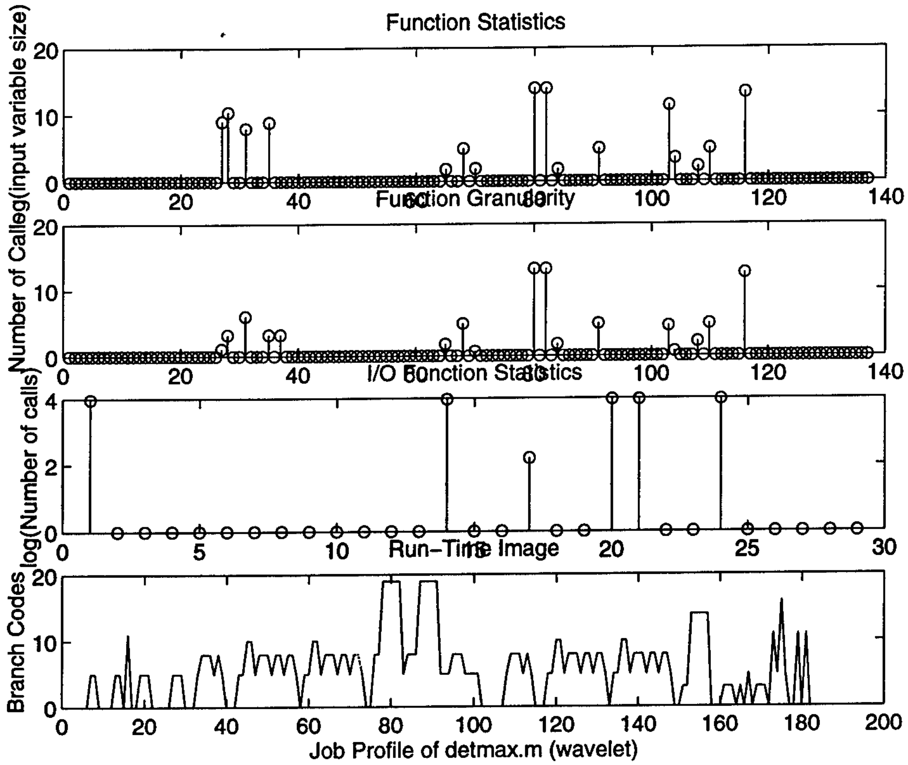

Figure 4.2 Example of job profile from the Osculant Job Profile Generator.

4.2 Result Analysis

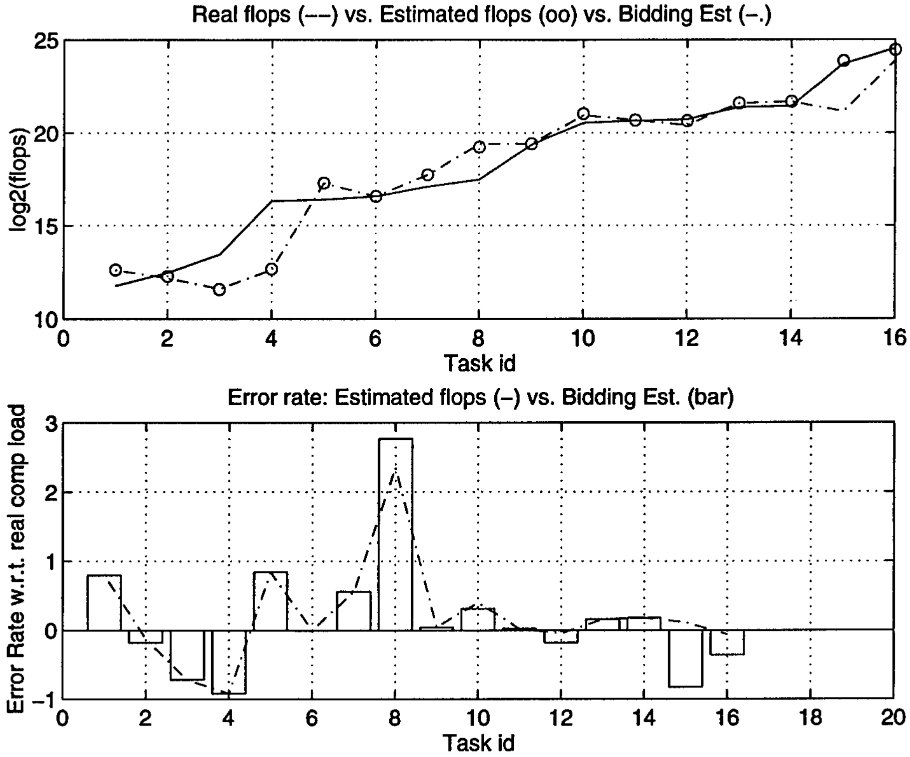

Figure 4.3 Estimation of computation load of 51 MATLAB sample jobs. The estimated number of floating point operations (flops) and bidding estimates are compared to the real computation load.

Figure 4.3 shows the results from the JPG on estimating computation loads for various tasks. Three types of results are shown in the figure, namely:

1. Real Flops: The real flops numbers are recorded after jobs are executed.
2. Estimated Flops: The estimated flops values are calculated by the JPG running at the user node. In this case, the generator knows the size and values of all input variables when functions are called. Some of the size and values of inputs are known or explicitly defined in the jobs such as user inputs and constants.
3. Bidding Estimates: Bidding estimates are calculated at the bidding node. Because it is impractical to include detailed information of input parameters for every function call

in the job profile, only granularity indices (see Section 4.1.2) are embedded in job profiles. Therefore, participating nodes need to estimate the computation load based on function statistics and granularity indices. It is less accurate than estimated flops from user nodes.

The program/job samples used in this studies contains 51 MATLAB user functions that are partitioned in the following categories:

1. One-dimensional digital filter designs in lattice, DCT, DHT, fixed-point arithmetic (81%),
2. Wavelet applications on speech recognition (6.2%), and
3. Two-dimensional digital filter designs and application on image processing (12.5%).

For most jobs, the error rate varies from 2% to 90% with respect to the actual computation load. The normalized error rate, which is found by normalizing errors with respect to real flops, gives a better indication for the overall quality of the Osculant JPG. The normalized error rate for estimated flops and bidding estimates are 10.78% and 45.46%, respectively. When it was recorded, these estimations are made without actually executing the programs. The error rates are considered acceptable for the purpose of preliminary or first-time bidding. Once tasks become familiar with system. A profile generator can be performed also by an artificial neural network (ANN).

4.2.1 Under Estimation of the Computation Load

In most cases, the JPG is optimistic in estimating computation resource requirements because of the worst case selection in the processes. Some job profiles generated by the JPG violate this principle because:

1. Some I/O functions contribute to computation loads. Some examples are spectrum plots and alignments;
2. JPG has limited capabilities in estimating the number of iterations in loops.
3. There are some problems in designing the Osculant Function Profile (OFP). Currently the OFP is simple and covers only a small range in input variables: The current version of OFP uses a 4^{th} order polynomial in estimating the computation load. When the input variable is beyond the pre-defined range, results may be flawed. Furthermore, some functions have separate algorithms for different input variable sizes (e.g., radix-2 and radix-4 FFT). A simple solution to this problem is to have separate profiles for this types of functions.

4.2.2 Over Estimation of the Computation Load

The profile generator intentionally produces an optimistic estimate on the computation load and job specifications. Optimistic estimates not only provide excess information of the tasks, but also is more suitable for real-time applications. Some improvements, such as a more sophisticated OFP design, may be considered so that larger input variable range can be covered.

4.3 Job Profile Retrospective

In an Osculant system, generating job profiles is the first step in the execution of a job. Typically, inaccurate job profiles normally will neither seriously degrade the overall system performance, nor produce incorrect results. Inaccuracy can be corrected or compensated for possibly in latter stages of job processing. For example, the bidding process itself can correct the errors in job profiles. Furthermore, Osculant scheme contains close-loop feedback that can correct errors as well. Figure 4.4 shows the life cycle of jobs in Osculant scheme based on the viewpoint of job profiles. Function category faults in generating profiles, for example, results in job rejections before job executions. Function profile inaccuracy results in degraded performance. These concerns can be easily corrected by adjusting the FRTT. A very sophisticated profile generator design, however, may not be worth the tradeoff for a decrease in generality and efficiency.

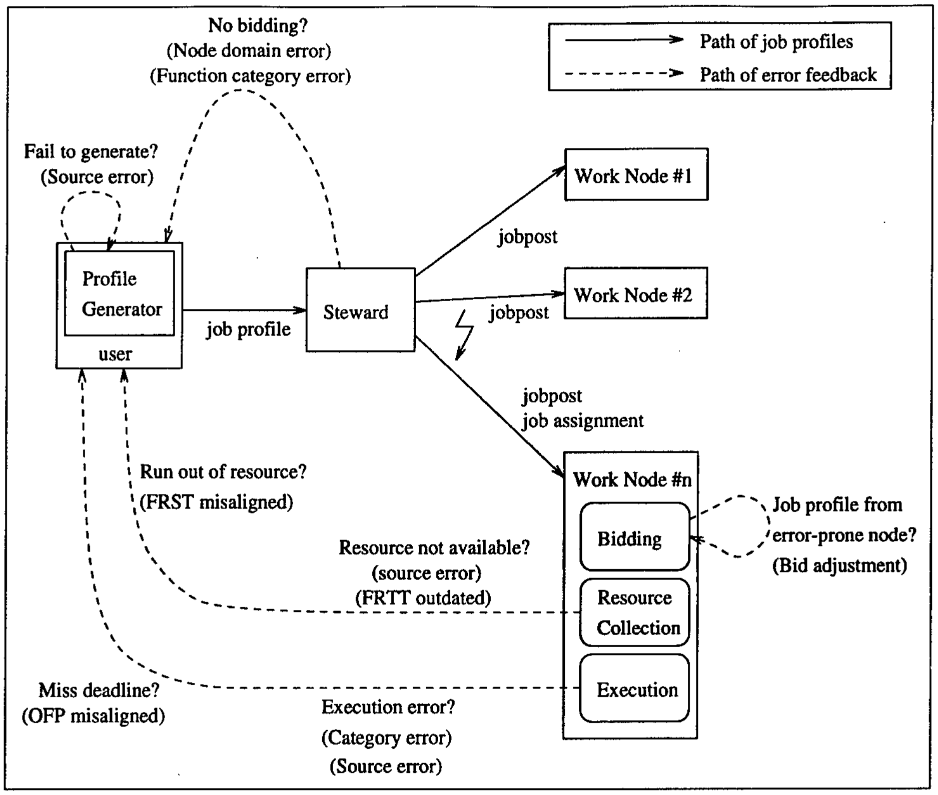

Figure 4.4 The figure shows the Osculant Job Profile Generator in the Osculant scheme. The accuracy and quality of job profiles can be fed back to the origin of the job profile from different stages of job executions.

4.4 Job Profile Generator Conclusions

The Osculant Job Profile Generator is designed to extract vital information from various tasks. The quality of job profiles greatly affects scheduling performance. The format of the resulting job profiles is defined so that job profiles can be utilized by various computation devices in a heterogeneous computing environment in order to estimate possible completion costs. Because of the heterogeneous and versatile nature of Osculant/network computing, the basic design of the profile generator is surprisingly simple and general. As a result, it can be adapted easily to other platforms and languages. The design of Function Result Type Table provides a capability to estimate results of functions in a flexible manner. Finally, results indicate that the estimates are accurate in the designed framework and possess a great potential for post-processing calibrations.

The application domain, to which the profile generator technology is applicable, is broad and includes real-time applications. Traditional hard real-time environments require all tasks to be profiled fully before scheduling them into the system. Historically, this limits the number of applications that can be accepted by a system and results in poor resource utilization. With job profiles, network computing systems is not only more efficient, but it also is more fault-tolerant because job profiles can be examined prior to actual executions. This technology can also be applied to a software engineering domain where the programming quality and productivity are verifiable.

5 Jobpost Distribution Protocol

Jobpost distribution protocols are designed to distribute to participating nodes small packets which contain job profiles, resource locations, and execution specifications. Major concerns in designing the protocol are topology independence and jobpost efficiency because:

1. the Osculant scheduler is designed for a distributed, heterogeneous computing environment; and
2. jobpost efficiency directly affects the capability of the scheduler to probe, to search, and to gather information in the system.

Two parameters control and indicate the performance of jobposting: jobpost constraint and jobpost coverage. The former limits the distance how far jobposts can go and controls the jobpost/bidding delays, where the later determines the range which jobposts reached in a system. It also represents the optimization that a job achieves in the system.

5.1 Limited Flooding Techniques

Flooding broadcast [Chow 1996] forms the basis of our jobpost distribution protocol. This technique guarantees that all nodes which meet the defined distribution rule will receive jobposts even when there are failures in the system:

Flooding Broadcast Distribution Rule: A node is either: *susceptible* (nodes never hears the jobpost) or *infectious* (nodes know the jobpost). When a susceptible node receives a jobpost, it becomes infectious and relays the jobpost to its neighbors. When an infectious node receives a jobpost that has been seen before, it does not react.

Multilayer Multicast Jobpost Protocol (MMJP): MMJP contains three major components:

1. A flooding broadcast is used to distribute jobposts in a single jobpost/bidding layer;
2. A jobpost constraint, in the unit of communication hubs, limits the range of jobpost distributions in a jobpost/bidding layer, and
3. In multilayer jobpost/bidding, the winner of current layer repeats the MMJP until a node wins in consecutive jobpost/bidding layers.

MMJP satisfies the goals of topology-independent jobposting, balancing and regulating jobpost/bidding delay, controlling the level of optimization, and self-organizing in the Osculant scheme. An example is shown in Figure 5.1.

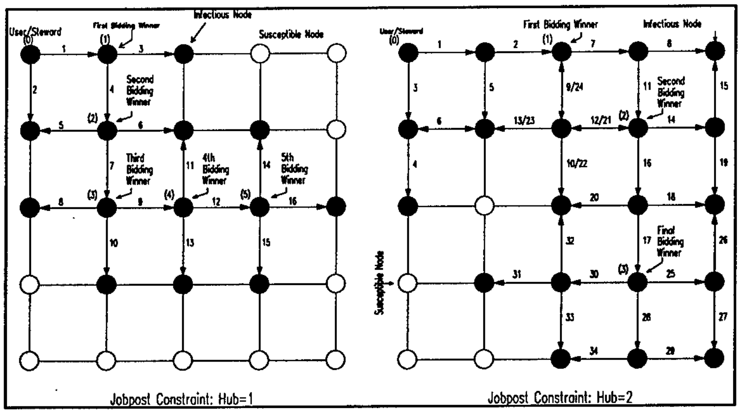

Figure 5.1 This figure shows an example of multilayer multicast jobpost/bidding. It indicates that 25 processors are connected by a mesh structure. With the jobpost constraint in the unit of hub number equals 1, as shown in the left figure, there are 5 levels of jobpost/bidding processes with a jobpost coverage of 60% and with 16 messages. If the constraint is increased to 2, it requires only 3 jobpost/bidding levels to have a jobpost coverage of 84% (21/25), and it needs to pass 34 messages.

Being an anomalous distribution protocol, MMJP is convergent, given that infectious nodes will neither re-post nor re-bid previous jobs. However, later studies remove this restraint (see Section 6). Convergence of MMJP then is enforced by bidding strategies.

Balancing between jobpost/bidding overheads and level of optimization is an important aim in designing a distributed scheduler. A flat jobpost structure, which has loose job constraints, generally has higher jobpost coverage and, therefore, produces more optimal results. But this structure also is more vulnerable to high jobpost/bidding overheads because of either failures in nodes or in communication channels. Moreover, there will be a greater message overhead burden (sending jobposts to *infectious* nodes). Conversely, distributions with restricted jobpost constraints normally have quicker jobpost/bidding process, but possess a more narrow scope in the system status. Consequently, such systems are inclined to be trapped in local optimums. Our studies indicate that small jobpost constraints (2 or 3 hubs) have sufficient jobpost coverage plus low overheads. Section 5.3 shows more results.

5.2 Other Jobpost Distribution Techniques

In some cases, such as in military applications, the number of messages traveled in the system must be minimized so that the probability of being detected/intercepted is reduced. In an computing environment where customers will be charged for using communication channels, it also is very desirable to reduce the number of messages in order to reduce cost. Therefore, by relaxing the requirement that all processors should receive jobposts, the number of messages transmitted in a system can be reduced. The candidates under studies contain epidemic algorithm [Chow 1996], or anti-entropy algorithm.

In most applications, it is not strictly required that all active processors receive jobposts and bid for jobs. Tasks can be completed as long as bidding participants can provide required system services. The number of bidding participants only represents the level of optimization that can be obtained. For the example in a network computing scenario with many resource suppliers in the system, it is not necessary that every active agent joins the jobpost/bidding process. Jobs still can be completed because resources are duplicated at many locations. The differences will be only cost and, possibly, service quality. Another distributed computing example is that accuracy of executing jobs can be guaranteed by setting appropriate read/write quorums to system functions. For example, suppose an updated system function or routine is required to be duplicated in more than half of the servers. When a user requests a job that calls this function, it needs only half of the servers in the system to participate in the jobpost/bidding process, and then the job can be guaranteed to be processed correctly.

5.3 Optimal Jobpost Distribution

The optimization criteria of jobpost distributions depends on the system configuration and job requirements. In this section, the criteria under investigation are message efficiency, jobpost coverage, and jobpost/bidding levels. The quality of a jobpost distribution method is determined by the ratio that it successfully reaches the best node, or the ratio of overall execution time from actual case to ideal case. The ideal case has a jobpost distribution with 100% jobpost coverage and a uniform jobpost/bidding delay from any two processors in the system.

The example shown in Figure 5.1 displays the effect of jobpost constraints on jobpost distribution quality. In this section, the jobpost distribution is studied on a mesh structure. Jobposts are distributed by using the flooding broadcast method with jobpost constraints in hub number. The bidding algorithm applied here is the performance based bidding method (Details in Section 6). Jobs are generated with the same processing time and I/O time, but with resources uniformly distributed in the system. The experiments show that:

- Jobpost coverage and message efficiency: As shown in Figure 5.2, the growth rate in the number of messages is greater than that of the jobpost coverage with increasing hub number constraints. As hub number increases, redundant messages transmitted in the system increases. The multi-layer flooding broadcast method only suppresses the number of nodes between the new and old *stewards* to multicast old jobposts, and this method cannot trace nodes that already knew the jobpost which is not on the path. The epidemic method will achieve better message efficiency performance than the flooding broadcast method.
- Number of jobpost/bidding layers: The number of jobpost/bidding layers decreases steadily as the hub number constraint increases (Figure 5.2(b)). Job auction process in a *steward* is hold once all bids are received or once a pre-defined time-out period is reached. In general, a time-out period will be used only when there are failures in the system. So, in the system is in a normal state, the auction delay is determined by the longest bidding delay between *steward* and the child nodes. If the logical structure of an Osculant system is close to the under-lying physical structure, i.e.

small hub numbers is equivalent to short bidding delays, more jobpost/bidding layers may have the advantage in better message efficiency while providing sufficient jobpost coverage.

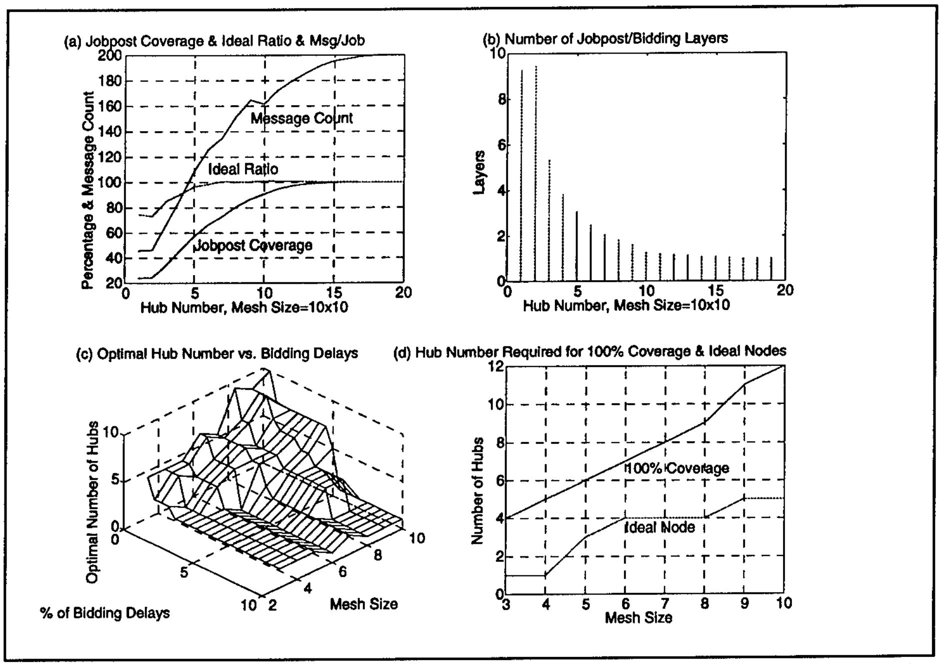

Figure 5.2 Simulation results and optimal jobpost constraints in hub number.

- Resource distribution: For jobs with fixed resource servers, or for systems with large centralized resource servers, it is optimal to have small hub number constraints because the cost surface is mostly monotonically decreasing toward the optimal node from any processor in the system. Small hub numbers will have highest message efficiency and the greatest scheduling quality.
- Optimal jobpost constraints in hub number: The optimal hub number can be achieved in two ways. First, a system with long bidding delays will prefer to have small hub number constraints. This relation is shown in Figure 5.2(c); the optimal hub number is generally a floor function of the bidding delays. For example, if the

bidding delay is between 1% to 4.5% of the average job processing time, the optimal hub number constraint is 7. If the bidding delay exceeds 5.5%, the optimal value will drop to 1. The second way is observed from the jobpost coverage and ideal ratios. Figure 5.2(d) indicates that the relation between the 100% jobpost coverage and the 100% ideal ratio is practically linear. It is sufficient to use a jobpost constraint, which is about 75% of the full jobpost coverage constraints, so that the optimal system performance can be achieved. Figure 5.2(a) shows the example that, in a 10x10 mesh, a hub number constraint of 5 is sufficient to reach the optimal system performance while the constraint for a 100% jobpost coverage is 12.

6 Bidding Strategies

As stated in Section 2, participating nodes submit bids which represent the status, intention or profit they can achieve from posting jobs. With additional considerations of the underlying computing environment, bidding processes are to be designed with the following purposes:

- Locally calculated bids: Participating nodes should calculate bids based on the information provided in the jobposts and system status available locally. The intention is to reduce the network traffic and to establish a loose, distributed and bottom-up scheduling style.
- Simple job auction process: The function of *steward* node is to distribute jobposts, to collect bids, and to designate winning nodes in an Osculant system. Stewards should be kept as simple as possible. As a virtue of bottom-up and distributed system design, vital information must be kept locally. Therefore, the failure of a *steward* node, or any part in the system, will not represent a severe threat. In addition, the designed computing system can be easily reconfigured by choosing other nodes to serve as a *steward* at any moment.

6.1 Performance-based Bidding Method

A performance-based bidding method is the most basic bidding method. The objective is to balance loads – both on processing units and on the network traffic connecting participating nodes. A good load-balanced system will generally have better performance because of reduced network congestion. With top-down scheduling schemes, load balancing can be achieved rather easily in a homogeneous, master-slave style computing system. Otherwise, load balancing is hard to achieve, given the difficulties in collecting local node status, and in being aware of non-scheduled local events. In this model, bids are calculated locally by participating nodes which reflect the cost of completing jobs. Three key components of the performance-based bidding method are as follows:

- Resource collection time estimation: An estimate of the resources, which is required to process jobs and may be distributed among the system, is needed to define a rational bid. Prior to job execution, the required resources must be verified and possibly transmitted over the network. The resource transmission time is governed by the size of resources, network traffic condition and cache storage status. Of the three factors, only resource sizes and cache availability is known locally and thus can be correctly estimated. Because of various network traffic condition, the transmission time will be estimated by the number of hubs between resource servers and bidders.
- Task execution time estimation: Local schedulers calculate the task execution time. For single-bidding strategies, a first-in-first-out (FIFO) scheduling scheme is employed. It is assumed that tasks are executed when all resources were received and previously assigned tasks were finished. Therefore,

$$\text{Task Execution Time} = \text{Max(I/O time, Completion Time of Last Task)} \\ + \text{(Task CPU Time)}$$

- Bid suppression: Bid suppression methods play an important role in the Osculant scheduler design. First, bid suppression is used to justify errors made in previously bidden jobs. Previous bidding errors are used to adjust bids applied to current jobs. Second, bid suppression can be used as a "buffer" or "cushion" to resist clusters of task assignments. Clustering task assignments occurs when tasks enter the system in blocks during a short time period. The reasons for clustering job assignments are:

(1) nodes do not update their states (CPU, I/O and network load) until they receive job assignments, and

(2) there are time lapses among jobposting, bidding, and receiving task assignments.

Therefore, job assignments for these tasks will be sent to the same node. The observed load diagrams for nodes without bid suppression generally have a "saw-tooth" shape. Short-term scheduling performance will also be degraded. Thus, the design of bid suppressor contains two factors, namely the previous bidding errors and the number of jobs which are bided but not be assigned.

$$\text{Suppressed Bid} = \text{Current Bid} * (1+ \text{Bidding\_Error})^{\text{MAX(Length of Job Queue, Number of Jobs Bided But Not Assigned)}}$$

6.2 Energy-based Bidding Method

This method assumes that when there is information transmission between two nodes, two parties will be required to be active for the same time period. Therefore, the overall energy consumed by the jobs is modeled to be:

Bid = (Task processing time) + (Estimated resource transmission time)*2

This is the simplest bidding algorithm in our studies since there is virtually no involvement of local or global scheduling in calculating the bids. Task processing time is derived from the job profiles. Similarly, resource transmission time is estimated from the cache status and raw communication bandwidth.

6.3 Dynamic Jobpost Model

Jobposts are continuously updated or modified during the jobpost/bidding phase. Ideally, the information stored in the jobposts becomes more localized as they approach the final working node. Also in this model, bids calculated by the participating nodes will become more and more "specific" because of less "estimating" on the global system status. This model also provides the capability of resource forwarding. Resource forwarding scheme grants the rights to servers which keep valid copies of resources to distribute the resources to others. In this design, original resource servers will no longer supply all the file services to other nodes, but perform more "selective" tasks, like providing new resources to the system, managerial tasks, cache validations, and resource distribution management. The work load and network traffic at resource nodes will be greatly reduced. Figure 6.1 illustrates the resource forwarding in the Osculant.

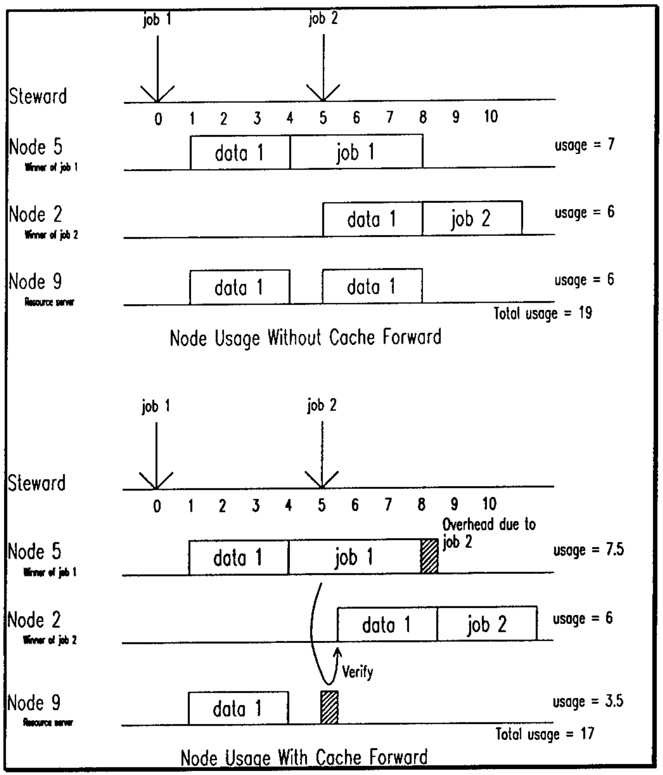

Figure 6.1 Example of resource forwarding in the Osculant.

The resource caching mechanism forms the basis of the this model. There are two bids that participating nodes need to calculate at the same time: the task processing bid and the resource supply bid. There is also two winners at each round of job auction process. The winner of the resource supply bidding updates the jobpost according to the local cache status and sends it to the task processing bidding winner. Then, the next round of jobpost/bidding or job execution proceeds. Cache verifications are performed before modifying the jobposts in order to ensure the correctness of jobposts.

Two jobpost and job auction methods are implemented. The following sections describe them in detail.

6.3.1 Static Jobpost and Job Auction Model

This model is the same as single-bid models. The winner of resource supply bidding and task processing bidding modifies the job profile. The winner of task processing bidding then distributes the new job profile to the next jobpost/bidding level.

6.3.2 Simulated Annealing Jobpost and Job Auction Model

Single-bidding and static jobpost method are problematic because final winners are mostly the local "best" bidders. Since the scope of bidding is limited by the jobpost constraints, job profiles commonly are bid in a few levels and trap in a local minimum node. Furthermore, the winning node and resources are determined statically as well as locally. As a result, the final winners of tasks and locations of resource duplications tend to form clusters. Clustering does not necessarily lower the system performance, but certainly reduces the reliability of the entire system.

The simulated annealing (SA) technique [Rutenbar 1989] is introduced to improve this situation. The choice of simulated annealing is based on the following points:

1. Global optimization with few restrictions: SA is capable of achieving global optimization by re-configuring the system structure. The initial system normally begins with a chaotic setup. Similarly, the states of nodes in a distributed computing environment are, generally, difficult to synchronize and to collect. Consequently, optimal scheduling is extremely difficult to achieve. Thus, the states of nodes can be treated as chaos as in most SA problems. While few restrictions exist in applying SA techniques, SA is considered suitable for various network topologies and node

configurations formed in a distributed computing environment.

2. Distributed control: SA process is governed by locally collected information and by the temperature of cooling schedule. There is neither global traffic nor overall re-configurations. The only global parameter in SA is temperature, which can be easily carried by jobposts.
3. Progressive processing: SA is a progressive optimization method whose results are retrievable at any time during optimization. The level of optimization is proportional to SA execution time. Additionally, the level of optimization can be controlled by altering the components in simulated annealing.

The four key components of simulated annealing algorithm are:

1. Move: Moves change system configurations. In Osculant, a move is defined as the sending of job assignments to the winning node.
2. Cost function: The purpose of a cost function is to calculate the differences between moves. A positive cost difference means that the system is moved to a higher energy state, and thus becomes more unstable. The cost function for the SA job auction process is the multiple-bid bidding algorithm.
3. Stop condition: Stop condition is a condition in which the *steward* node is the winner of the current jobpost/bidding level and all tried moves are failed.
4. Cooling schedule: The cooling schedule gradually lowers the temperature so that the system configuration becomes more stable in time. The temperature is:

$$\text{Temperature} = (1 - \text{Jobpost Coverage})/F(\text{Number of Successful Moves})$$

where F is an incremental function and successful moves are moves that lower the costs.

```
Job_Auction_Simulated_Annealing(job, time)
{   Collect_bids();
    MinBid = min(bids);
    Default_Winner = Node_with_MinBid;

    if (Steward~=Default_Winner)
    {   Send_Job_Assignment(Node_with_MinBid);
        Prev_Minbid = MinBid;
        Num_Succ_Move++;
    }
    else
    {   flag = 0;
        for (Node.Bid>MinBid)
        {   Cost_Diff = C(Node.bid - MinBid);
            Temp = (1 - Job_Coverage)/F(Num_Succ_Move);
            SA_Threshold = exp(-1*Cost_Diff/Temp);
            if (rand()<SA_Threshold)
            {   Prev_MinBid = Node.bid;
                Send_Job_Assignment(Node_with_MibBid);
                flag = 1;
            }
        }
        if (falg==0)
            Assign_Job(Steward);
    }
}
```

Figure 6.2 Simulated annealing job auction process.

Figure 6.2 shows the pseudo code for simulated annealing job auction process. The current version of function F is $\log_2$, and function C is $\log_{10}$.

6.4 Resource Contractor Bidding Model

Similar to the dynamic jobpost model, there are two bidding stages. The winning nodes in the resource supply bidding become the resource contractors which take the responsibility of collecting resources for jobs. Upon collecting resources, the contractor node forwards them to the winner of task processing bidding. The motivation behind this model is that the winning resource supply nodes usually have more local resources and also have better channels (high bandwidth) to access remote resources. For the task processing bidders, they solely need to estimate the network traffic condition toward the resource contractor and the local job completion costs. These arrangements will result in better bidding accuracy and local scheduling capability. This model also better suits the conventional configurations of local area network (LAN) connected nodes. In an LAN configuration, there are generally only a few nodes which serve as local resource servers with better network and file service performance. Under the contractor bidding model, these local servers autonomously become resource contractors and satisfy most local needs. Additionally, failures in local resource servers will only degrade performance (resources will host remotely or migrate to less-capable nodes in the LAN), but do not halt services as in conventional systems. Furthermore, resource distribution management can be conveniently performed in the resource supply bidding phase.

The resource contractor strategy is implemented by the two-stage bidding model plus a cache information exchange session. They are described as follows:

- Resource supply bidding: Participating nodes calculate bids which represent the cost of collecting all required resources. The single-bidding job auction model is applied in this stage. Winners of this stage become the *steward* of second stage bidding. The final winners provide information in the jobposts of second stage bidding about:

  (1) the identification of the resource supplier node,

  (2) the expected completion time to collect all resources, and

  (3) the estimated I/O load after the completion of collecting resources.

- Task processing bidding: Participating nodes calculate the task processing bids by using information provided by the first stage winner. Because contractor nodes provide estimated resource waiting time, all required resources will be transmitted from only one node. Normally, the contractor node locates locally. More accurate and aggressive local scheduling schemes can be applied. In the current implementation, task processing bidding is held immediately after the assignment of resource contractor node. Bidders calculate their bids according to the current cache status (speculative estimation; no cache verification), traffic condition toward the contractor node, and local CPU schedule.

- Cache information exchange: In order to reduce communication overhead between the resource contractor and task processing node, a cache information exchange session is performed right after the contractor node retrieves all resources from various servers. A packet with job resource list and current version number will be sent to the task process node in order to determine the types of resources to be transmitted.
- Out-of-order Local Scheduling (O3LS): O3LS becomes a possible solution to further improve system performance after the realization of two-stage bidding scheme. Contractor bidding model provides better bounded information about the availability of future expected resources. Therefore, local nodes can make better estimates and utilize the system's resources better. O3LS is currently applied on the local CPU time planning with non-preemptive scheduling techniques.

6.5 Comparisons Among The Bidding Strategies

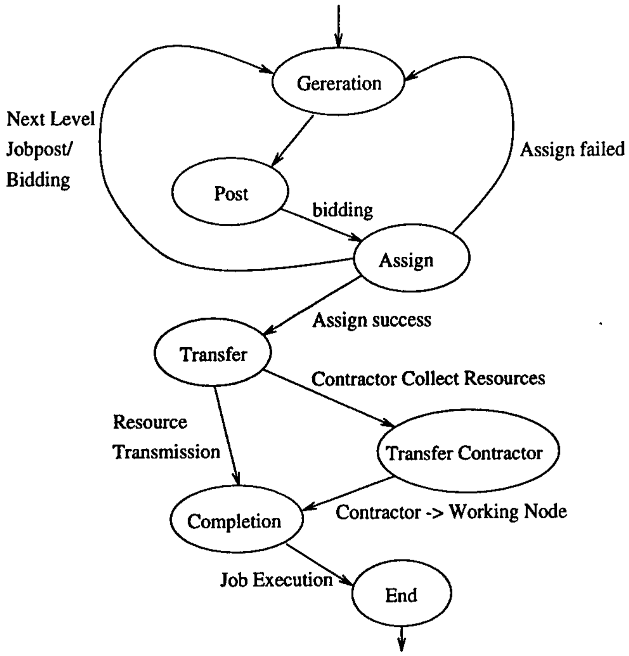

Figure 6.3 State diagram of jobs in Osculant.

Experiments were conducted using the Osculant Simulator (see Section 8) with the job state diagram shown in Figure 6.3. The results shown in this session is retrieved by using the configuration in Appendix A.

6.5.1 System Throughput Rate

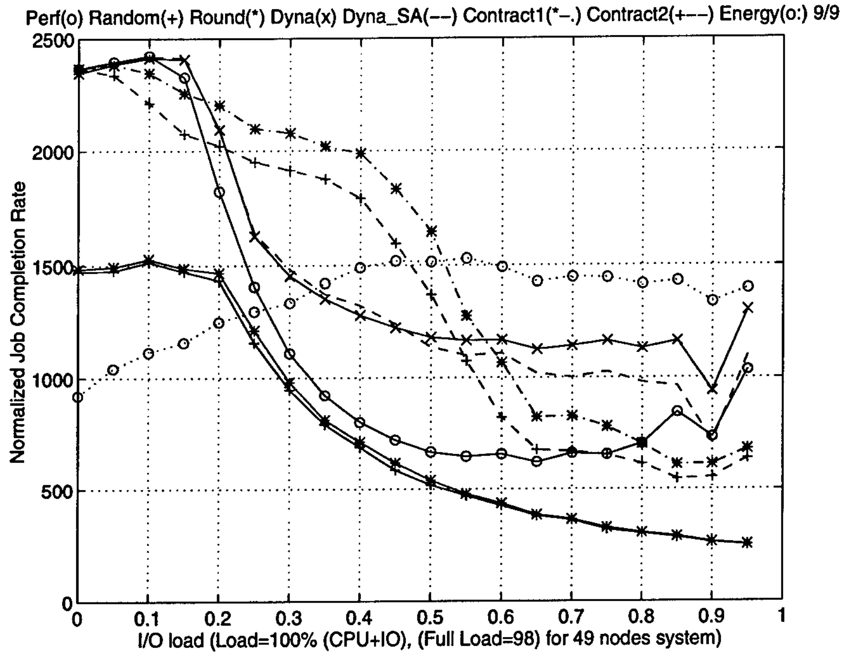

Figure 6.4 System throughput rate results.

System throughput rate (Fig 6.4) illustrates the capability of the scheduling scheme to utilize the system's resources. Various bidding models show their strength and weakness in different system configuration and job combinations. In general, multiple-bidding models outpace single-bidding methods, especially in a CPU-intensive computing environment. Dynamic jobpost models have the best throughput rate if average I/O load of tasks range below 10%. Resource contractor model performs well over job I/O ratios ranging from 10% to 50%. The advantages of contractor bidding scheme gradually gives way to dynamic jobpost methods, or performance-based bidding methods, as more I/O intensive jobs enter the system. Weak performance of contractor model in the low and high I/O load regions comes from excessive bidding stage and resource transmission session. In an environment where timing and task execution sequencing are enforced, the complexity of implementing contractor model will be of concern. With average I/O load exceeds 50%, energy-based bidding method behaves surprisingly well. Tasks are mostly assigned to nodes near resource servers. Depending on job composition, the overall performance of these bidding methods varies. Assuming distributions of job generation locations and compositions (in CPU time and I/O load) are uniform, system throughput rate improved by 74%, 147%, 125% and 177% over the random method for performance-based models, dynamic jobpost models, resource contractor models, and energy-based bidding models, respectively.

6.5.2 Average CPU Time Consumption

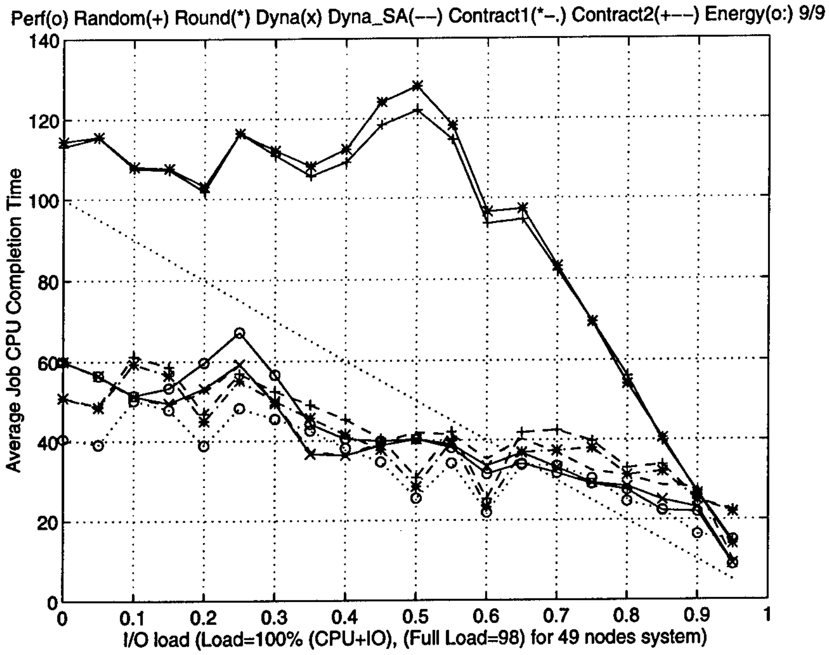

Figure 6.5 Average task CPU time consumption.

The results in Figure 6.5 show the adaptability of bidding schemes. The sample jobs have mean CPU processing time ranging from 100 (no I/O) to 5 (95% I/O time) time units (shown in dotted line in Figure 6.5). The nodes, on the other hand, have a mean CPU processing power of unity. The bidding schemes are efficient in scheduling jobs to high-computing power nodes. Furthermore, bidding schemes in Osculant are immune from performance degradation caused by system configuration alterations and by load fluctuations in top-down scheduling methods.

6.5.3 Average Job Resource Transmission Time

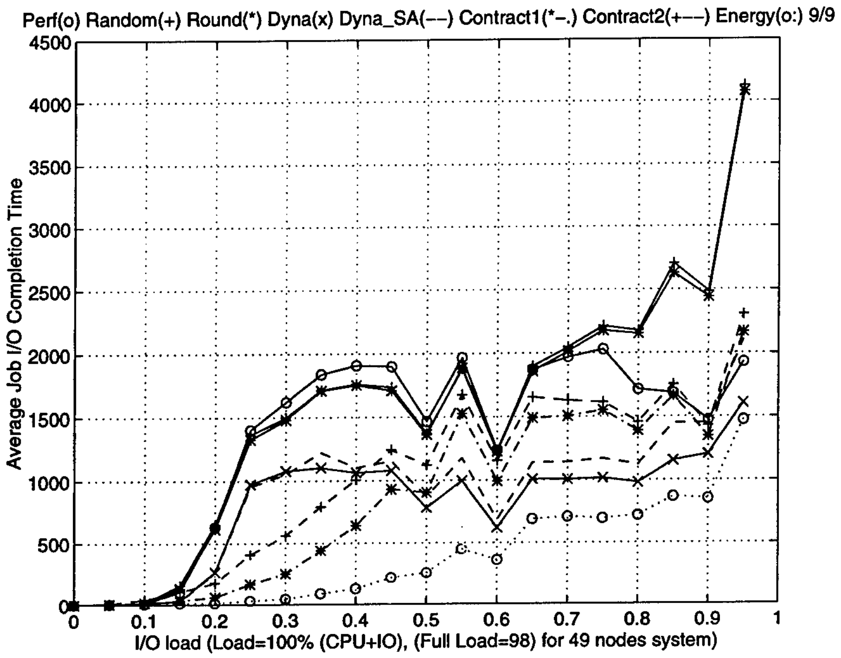

Figure 6.6 Average task resource transmission time.

Communication time of tasks is a major factor in determining the overall performance of a distributed computing system. The system throughput performance of various bidding models is mostly determined by transmission time. As illustrated in Figure 6.6, energy-based bidding scheme has the lowest I/O transmission time, although its job assignment distribution performance is poor in low I/O rate region. Resource contractor model and dynamic bidding model have a significantly lower transmission time than performance bidding model. Interestingly enough, most of the bidding schemes achieve a constant resource transmission time during a broad I/O ratio range. Increases in network transmission time are counter-balanced by efficient task assignments and by resource distributions in the Osculant.

6.5.4 Average Job Energy Consumption

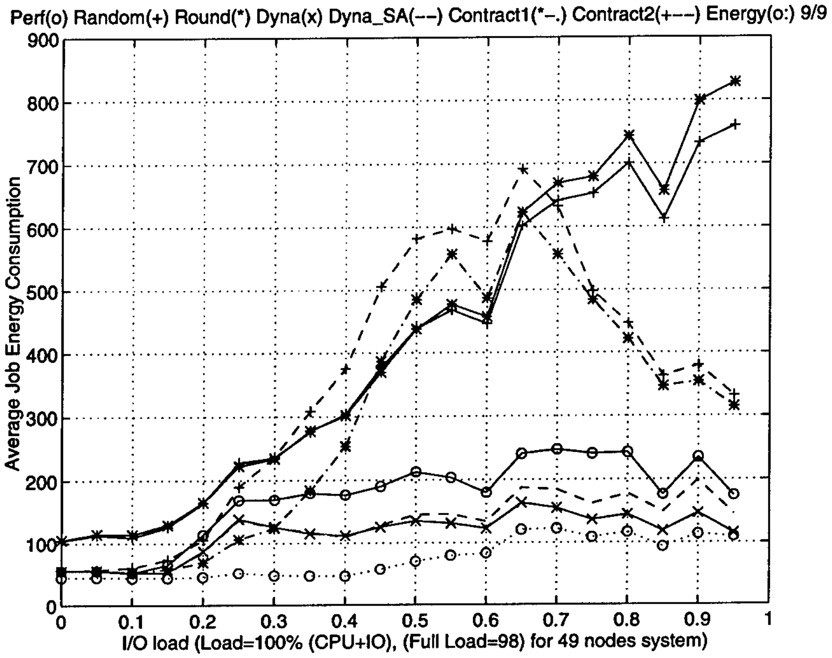

Figure 6.7 Average job energy consumption.

Energy consumption rate (shown in Figure 6.7) is defined as the time period that nodes need to be active because of assigned jobs. Overlaps in the communication time of newly assigned jobs and the processing time of previous jobs contribute to energy savings. Compared to system throughput rate, these results are very different among various bidding schemes. The contractor bidding method shows a relatively high average energy consumption rate because of its two resource transmission sessions. Overlaps in the resource contractor session are relatively small because the contractor rarely receives task processing assignments. In contrast, the energy-based bidding model demonstrates the lowest energy consumption rate for almost all ranges. In summary, average job energy consumption reduction from the bidding methods are significant. With the same job distribution in the throughput rate section, completed jobs utilize 48%, 35%, 70% and

22% of power consumed by jobs in the random method for the performance-based scheme, dynamic jobpost scheme, resource contractor scheme, and energy-based bidding scheme, respectively.

6.5.5 Average Jobpost Coverage and Jobpost/bidding Delay

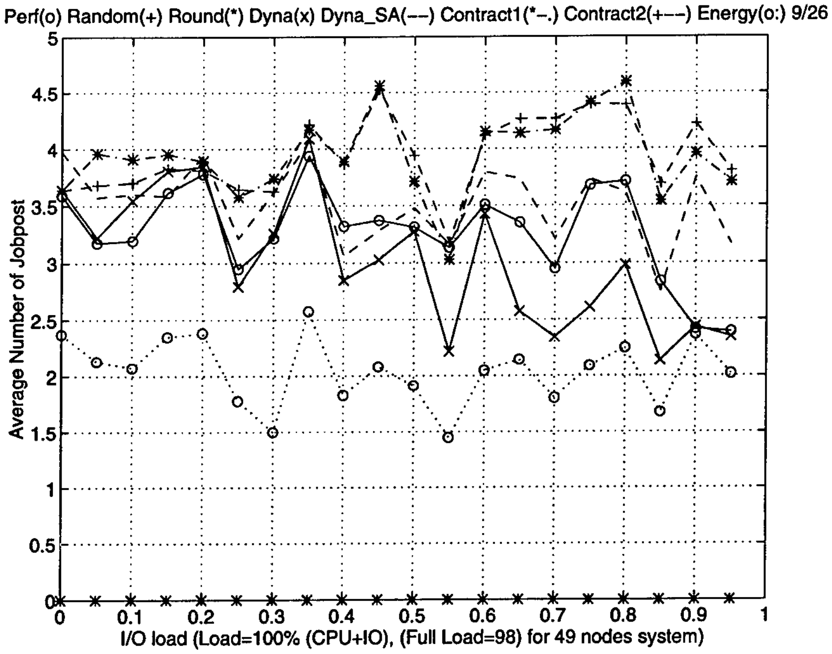

Figure 6.8 Level of jobpost/bidding.

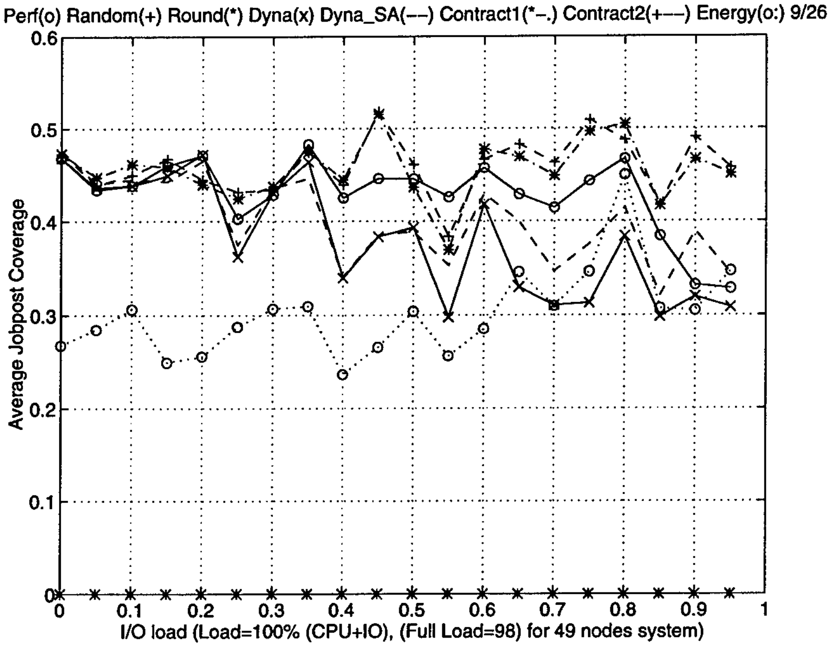

Figure 6.9 Jobpost coverage.

As discussed in Section 5, jobpost constraints and the number of jobpost/bidding levels control the degree of optimizations that jobs can find in the system. This statement is argumentatively correct for single bidding schemes. These experiments show that resource contractor model has the highest jobpost coverage and the greatest level of jobpost-bidding because of the two stages of bidding involved in this model. Dynamic jobpost models have relatively high number in jobpost/bidding levels, but a low jobpost coverage, which suggests the effectiveness of dynamic jobpost modifications in this model. Unfortunately, dynamic jobpost modifications also increase the possibility of being trapped in a local sub-optimal location. This correlation may explain the unsatisfied performance of this model in the mid-high job I/O load range. The energy-based bidding model has a surprisingly low number in jobpost/bidding levels, which results in low bidding delay, and moderate jobpost coverage performed best in mid-high job I/O load range. In the I/O intensive environment, jobs are better assigned to resource servers, or to their neighborhood, in order to reduce the communication overhead. This is the role of the energy-based bidding model. The simulation results are shown in Figure 6.8 and 6.9.

6.5.6 Cache Efficiency

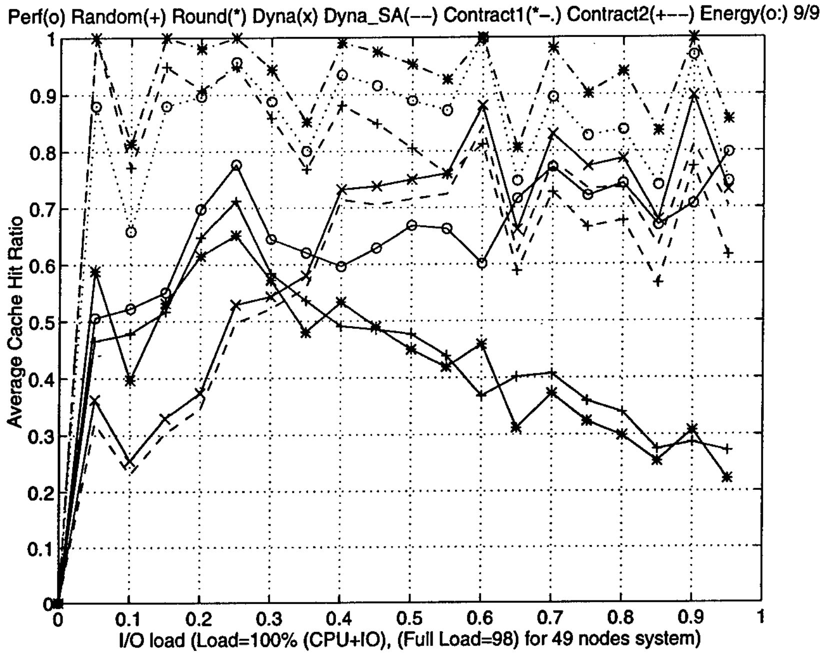

Figure 6.10 Cache efficiencies.

Experimental results in cache efficiencies (Figure 6.10) suggest the advantages of contractor bidding and of energy-based bidding models. In the Osculant, the transient resource allocations are autonomous and are driven by demands. Thus, they are flexible and adaptive. Resources which are vital to system operations, or are in high demands, can be hosted in more nodes. Otherwise, a minimum number of duplications are maintained to reserve storage space. Moreover, controlling the number of duplications improves the variety of replicated resources, which contributes to the reliability of the system.

7 Resource Management Schemes

According to Goscinski [Goscinski 1991], resources are reusable and relatively stable hardware or software components of a computer system that are useful to system users or to their processes. Because of their usefulness, they are requested, used, and released by processes during their activities. Resources also can be grouped as low-level resources and high-level resources. Low-level resources can be used directly by the distributed system and users. However, high-level resources, which are built upon several low level resources, are more commonly utilized in the system. Hardware resources are generally static, permanent, and limited in quantity. Management of physical resources are comparatively simpler than logical resources because the latter may be varied temporally and quantitatively.

Software resources can be pre-defined by the system or composed by the users. They can also be active (so that they can change states), or static. Furthermore, advances in network computing and in software engineering bring both new opportunities and problems to the management of logical resources. For instance, software components will be required to be properly managed in a heterogeneous system with differences in suitability for the underlying platforms and in service quality. In other words, multiple software versions and locations may exist that that the can be used and/or retrieved to complete the task with differing costs and functionality. Additionally, as purchasing and maintenance of software becomes more dominant in the overall operation cost of computing systems, licensing and revision control will become an important topic in the future resource management.

Software industry has begun to adapt "thin clients", "component software" and, "just-in-time applications" concepts. "Fat clients" are difficult to manage, and they add to network congestion. On the other hand, in the "thin clients" environment, applications are stored and managed centrally on servers. Appropriate executables and data files (logical resources) are sent and stored in local caches when needed. Additionally, some observations justify "thin clients" and "just-in-time download" concepts:

(1) 80% of users access only 20% features which are implemented in an application, and

(2) most applications can start with a relatively small portion of the whole packages.

Castanet [Thomas 1997], by Marimba Corp., and *ALTiS* [Goulde 1997], by EPiCON Corp., are two pioneer systems built on the thin client concepts to distribute Java applications. Corel Corp. tried and demonstrated some of the WordPerfect suite (*Corel Office JV*) in their JAVA implementation so that applications can be launched and executed from web browsers on different platforms. *ComponentWare* [ComponentWare 1997], by I-Kinetics Inc., is a design based on Common Object Request Broker Architecture (COBRA) that integrates customer applications from pre-fabricated software components. These trends suggest that software resources are changing. Existing scheduling and resource management methods may be inefficient to handle them.

7.1 Resource Forwarding and Caching

The current development of the Osculant scheduler is to introduce a new and integrated resource management scheme to further improve the system performance. It is evident that proper resource allocations are essential to improve system performance. Moreover, while more copies of same resources improves the performance of some tasks, it is also desired to have more types of resources duplicated and distributed so that the designed system can be more robust and balanced. In Osculant, the resource distribution scheme is implemented by various file caching and forwarding techniques. The resource management scheme is driven in a bottom-up manner and grants the responsibility of resource allocation, cache coherence maintenance, and distribution pattern enforcement.

7.2 Resource Distribution Control via Cache Validations

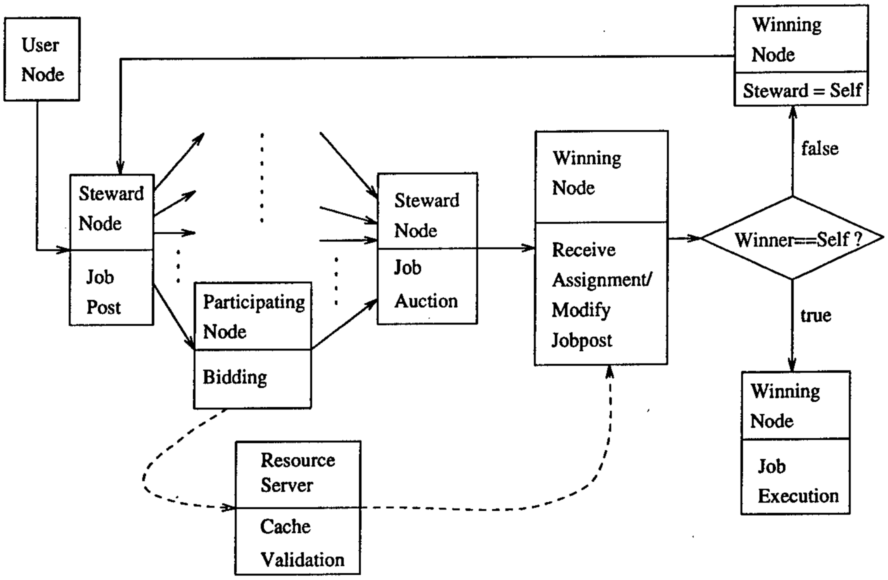

Figure 7.1 Cache validation process in the Osculant scheme.

Cache validation processes used by the Osculant is shown in Figure 7.1. Nodes will issue cache verification requests to resource servers upon receiving jobposts. In order to reduce the bidding delays, bids are calculated before knowing the verification results (speculative bidding). It is possible that a node receives job assignments based on expired cached copies. This type of error is treated as bidding errors and generally does not repeat itself in the next bidding. Bidding errors do not affect the correctness of job executions because the cache verification results are required prior to the job execution phase. Normally, cache validation process is completed earlier than the combined time length of bidding and job auction processes. After integrating the cache verification and bidding processes, there will be no extra overhead introduced in this section.

Currently, two resource management models are implemented:

- Plain Model: Based on First-In-First-Out (FIFO) principle, when a new node holds a copy of the resource, the oldest node in the list is flushed. From the experiments, it is shown that:
  1. This method is simple and introduces the lowest network traffic load for cache message exchanges.
  2. Trashing affects the performance because useful resources might be replaced or flushed by any newly requested resources.
- Request Frequency Model: In this model, updates of resource entries are based on the frequency of validation requests. Resource servers maintain several counters which record the number of validation request from other nodes. The counter also decreases periodically according to the real-time clock. Simulation results indicate that this method is more efficient in reducing network traffic among nodes. Preliminary results are shown in Table 7.2. However, this model cannot control the geology distribution of resources.

Improvement	Perf	DynaJP	DynaJP SA	ResCont1	ResCont2	Energy
Average CPU Time	-0.0882	-0.1886	-0.1863	-0.0234	-0.0144	-0.1705
Average I/O Time	-0.2000	-0.4273	-0.3843	-0.4655	-0.3202	-0.6296
Energy Consumption	-0.2033	-0.3971	-0.3933	-0.4681	-0.3496	-0.4309
Cache Hit Rate	2.8018	1.3488	1.5173	0.6177	0.8525	0.4181
Throughput Rate	0.3228	0.6848	0.6818	1.2414	0.8620	0.6377
I/O Ridding Error	-0.0734	-0.1207	-0.1413	-0.0968	-0.0454	-0.1180
Processing Time Error	-0.1502	0.8488	0.8320	-0.1863	-0.1200	-0.5517
Overall Utilization	0.0557	0.0365	0.0365	-0.2077	-0.1616	-0.3921
Server Utilization	-0.0610	-0.1007	-0.0927	-0.1738	-0.1247	-0.2660

Table 7.2 Performance comparisons between the Plain Model and Request Frequency Model.

8 Osculant Simulator

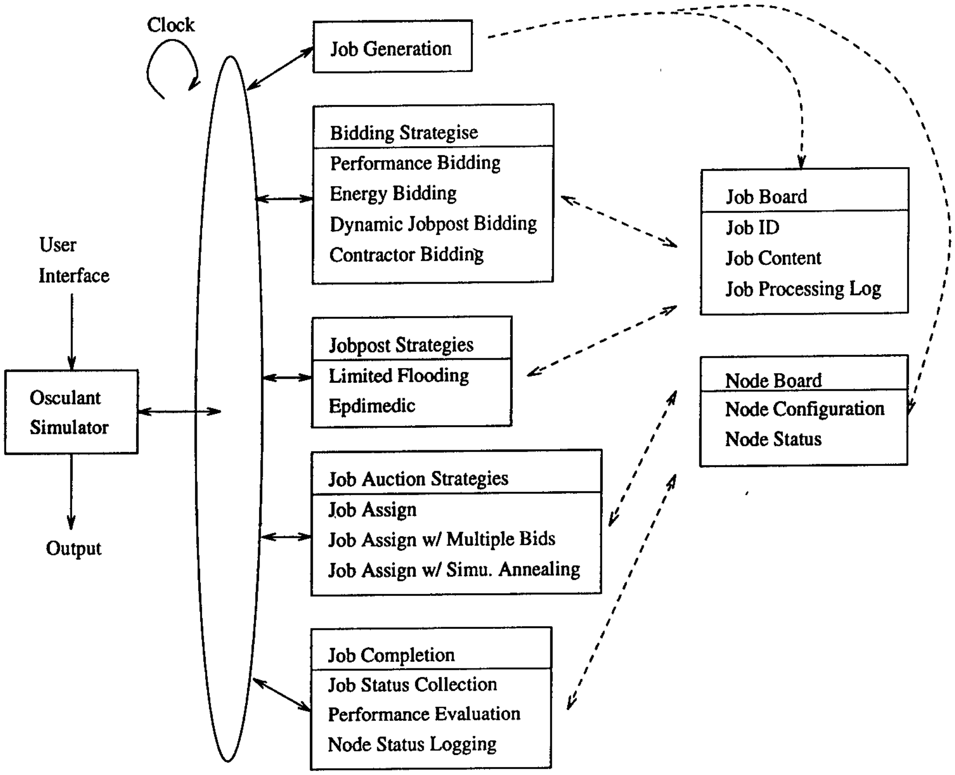

Figure 8.1 Structure of the Osculant Simulator.

The Osculant Simulator is designed to study the Osculant scheduling scheme by simulating various types of computing environment and external conditions. The current version of Osculant Simulator is capable of simulating job generation, dynamic jobpost, bidding, and performance evaluation.

The structure of the Osculant Simulator is illustrated in Figure 8.1. There are five major modules in the simulator: job generation, bidding strategies, jobpost strategies, job auction strategies, and job completion module. Each module contains a selection of algorithms and subroutines that operate each module. Each module accesses the Job Board and the Node Board for job contents, node configuration, and current system status.

The Osculant Simulator is implemented with the time-advance concept which scans all modules for events at each time step. Initial events, which contain job generation time, locations, resource requirements, and resource locations, node configurations, and network bandwidth, are generated using a MATLAB program with user-defined statistical models. Internal states of jobs and nodes are created and are inserted by the simulator during run-time. State diagram of jobs is shown in Figure 6.3.

The current version of Osculant Simulator is implemented by using C language and is executed on UNIX environments (tested on SUN OS, HP-UX, and LINUX). Appendix G illustrates the Osculant Simulator user interface. The user interface allows users to change a wide range of system parameters such as:

- job contents,
- node configurations,
- network configurations,
- bidding strategies,
- jobpost protocols,
- resource management schemes, and
- system parameters.

9 Osculant Shell

Osculant scheduling studies were conducted by simulations, and implemented using a custom UNIX Osculant Shell. The Shell connects, monitors, bids, and distributes MATLAB jobs and executable objects among a collection of HP and SUN workstations. The studies also utilized a custom Osculant Job Profile Generator implemented in MATLAB script and C language. These two software systems define the Osculant experimental environment.

9.1 Structure and Implementation of Osculant Shell

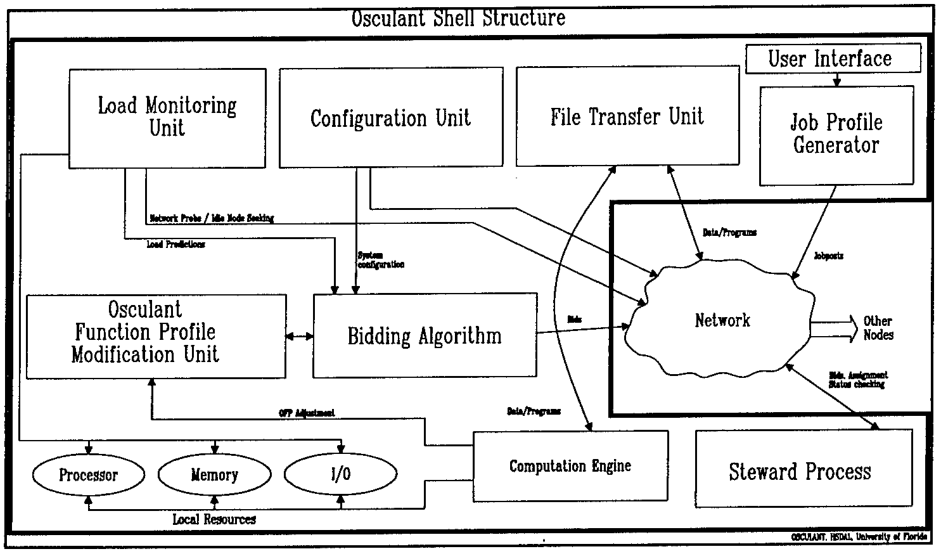

Fig 9.1 Structure of Osculant Structure.

Figure 9.1 shows the structure of Osculant Shell. The Osculant Shell consists a collection of modules interconnected by various communication channels. Modules independently process information that run simultaneously in the background. The shell is implemented in C language and Berkeley Sockets. The design of the Osculant environment is based on the consideration of portability and compatibility among other UNIX systems. The functions of modules is described below:

9.1.1 Osculant Job Profile Generator

The Osculant Job Profile Generator (JPG) produces job profiles for a task at the user's node. Job profiles are generated when users submit jobs for processing. JPG generates job profiles at run-time. Details of JPG are presented in Section 4.

9.1.2 File Transfer Unit

The File Transfer Unit (FTU) transfers data among nodes. In Osculant, data and job files can be distributed within several different nodes and transmissions can occur at any time. Performing multiple file transmission sessions will improve the performance of and the efficiency of communications. The FTU generates an individual process for each file transmission session.

File transmissions can be either active or passive, depending on what initiates the file transmission. In the active file transmission mode, the assigned node sets up the communication channels to the resource (e.g. data and job files) holders and "retrieves" the required data. This model is simple and fast, but is considered impractical and insecure. On the other hand, in the passive mode, a node receives a job assignment, and sends the request to resource holders, and then the resource holder initiates the transmission. Data is "sent" or "pushed" to the assigned node. The passive model requires one extra message passing stage but is more secure and practical than the first method because the resource holders are mostly the file servers of a system.

Messages transmitted among nodes can take one of the following forms:

- jobpost,
- job assignment,
- acknowledgement,
- status checking,
- bid,
- user input,
- data, and
- job files.

In order to correctly and effectively operate the Osculant scheduler, messages and data will must be handled in a proper order. An FTU module has two message receiving and message transmission queues. One pair of receiving/transmission queues are dedicated to the transmission of data and job files as first-in-first-out (FIFO) queues. The other message queues are priority queues. The priorities of messages are:

(1) Status Request (highest priority)

(2) Status Reply

(3) Job Completion

(4) Job Deletion

(5) Job Assignment Acknowledgement

(6) Job Assignment

(7) Bid

(8) Jobpost

(9) User Input (lowest priority)

9.1.3 Configuration Unit

Configuration Unit (CU) dynamically changes the system configuration based on the system status or user demands. It also provides a transparent view of the system to the user. The CU, however, is not implemented in the early version of Osculant Shell. Main functions of CU contain:

- Connect/disconnect nodes: Disconnect failed nodes in order to reduce bidding delays and to re-connect recovered nodes to the system.
- Add/delete nodes: Accept new nodes to the system or delete nodes from participating list.
- Calibrate system parameters.

9.1.4 Load Monitoring Unit

The Load Monitoring Unit (LMU) provides local node information for the purpose of bid generation. LMU adjusts the local node performance parameters to adapt to the local load requirements set by the local owners as if these nodes are privately owned. The module is not implemented in the early version of Osculant Shell. The main functions of this LMU are:

- Modified "Ping" function, which probes the network status.

- Idle node hunter which estimates the computation resource of a subsystem.
- Algorithmtic estimates of communication channel bandwidth and throughput stability.

9.1.5 Osculant Function Profile Modification Unit

As described in Section 4, the Osculant Function Profile (OFP) stores information about system functions and provides the basis for calculating bids. The module modifies the contents of OFP in order to improve the bidding accuracy after completing all assigned jobs. Details of modification algorithms are discussed in Section 4.

9.1.6 Bidding Algorithms

The bidding module utilizes information from other support units and calculates bids for each job. Various bidding strategies have been studied and are reported in Section 6.

9.1.7 Computation Engine

The computation engine of current version of Osculant Shell is MATLAB. The cooperation of Osculant Shell and MATLAB is established by using MATLAB External Interfaces [MATLAB 1993]. The External Interface Library is MATLAB's application programming interface (API) and is called from the C language within the Osculant Shell. The interface routines will create two pipes so that data and commands are transmitted and executed by the MATLAB engine. The MATLAB External Interface Library provides a platform for heterogeneous computing environment while running on SUN, HP, and other systems. The current version of Osculant Shell also accepts executable binary objects to be scheduled and executed by the system. These jobs can, however, only be executed in homogeneous systems.

9.1.8 Steward Process

Since processors perform their bidding autonomously, and the *steward* has the full authority to award job assignments to any working node based on bids, some top-down capabilities can be implemented in the Osculant scheduler to achieve certain goals like fast job assignments or improving the overall scheduling performance. In Osculant, the *steward* performs the following functions: job auction, status checking, fault handling and node training.

Job auction processes are discussed in Section 6. The *steward* also performs status checking for the system. Normally, the *steward* has responsibility to check the status of its child nodes. Faults are handled by redoing jobpost/bidding process among active neighboring processors. The Osculant scheduler also provides top-down control over the system in training and monitoring of working nodes. The *steward* can log and evaluate the bidding performance of its neighbors. Top-down training can be accomplished by duplicating job assignments and by sending them to the node with best bid and to the nodes that need further training. Therefore, training nodes can have more chances to calibrate their bidding components.

An additional feature of *steward* is its ability to perform reliable job processing. To perform reliable computing for an unreliable distributed system, the *steward* can duplicate job assignments and send them to working nodes without sharing resources. Failures caused by crackdowns in any single resources, therefore, can be greatly reduced. Compared to other fault tolerance schemes, this method is the simplest to implement. This method also provides higher guarantee level in processing critical tasks in a real-time system.

9.2 Future Developments

The current version of the Osculant Shell is used for evaluation and demonstration purposes. The scheduling overhead is considered to be too high for serious commercial applications. To move Osculant to the next level, some key developments and improvements are needed. They are described below:

- The current version of Osculant Shell is implemented at the user level. This results in many limitations that inhibit the scope and the capability of the scheduler. For instance, the shell can neither directly and efficiently observe the system status nor change or control the processes running foreground or background. Figure 9.2 shows the position of current version of Osculant Shell.

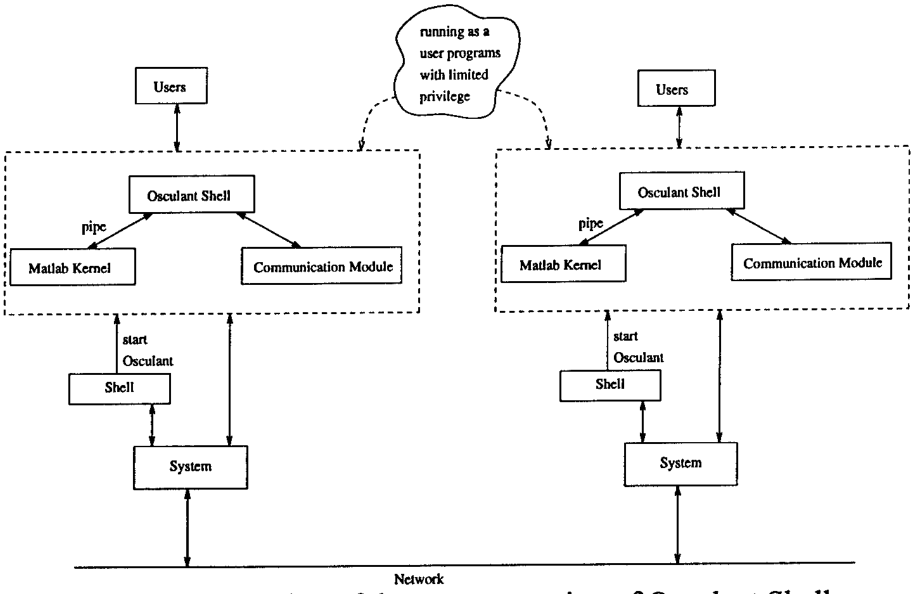

Figure 9.2 Location of the current version of Osculant Shell.

- The process-based Osculant Shell is currently inefficient because it creates processes in the UNIX environment which consumes a substantial amount of resources in memory and CPU time. Furthermore, communications among processes remain inefficient because the current Osculant Shell uses message passing based protocols and mechanisms. It is better for the Osculant Shell to be implemented by thread-based structure.

The repositioning of the Osculant Shell in an operation system requires additional planning: As mentioned above, the shell should be located in a lower level of a system to improve the performance. However, implementing the Osculant Shell in a very low level reduces the possibility of porting it to other heterogeneous systems. To implement the Osculant Shell in a way that balances the need for performance and for portability, a good location for the Osculant Shell is between the command shell and the system kernel.

10 Future Development of Osculant Scheduler

10.1 Task Organizer and Optimal Scheduling

In a computing system, resources are mostly requested in the high level format, e.g., system services comprised several low level resources. In general, end-users or system processes have simple and abstract "goals" with regards how they schedule their jobs. Scheduling in the context of these "simple" goals is straightforward, but not likely to be optimal in performance. Furthermore, trends change in the future infrastructure of computing environment (e.g., network computing, modulated software applications, execution while downloading, and multiple agents), task scheduling and resource management should evolve themselves with considerations of these ideas.

Motivations for such studies are based on the observation that task execution constraints can be refined, determined or even defined on the scheduling (or execution) time. As found in previous Osculant studies, it is possible that a systematic and autonomous scheduling scheme can be developed so that the task completion cost can be minimized.

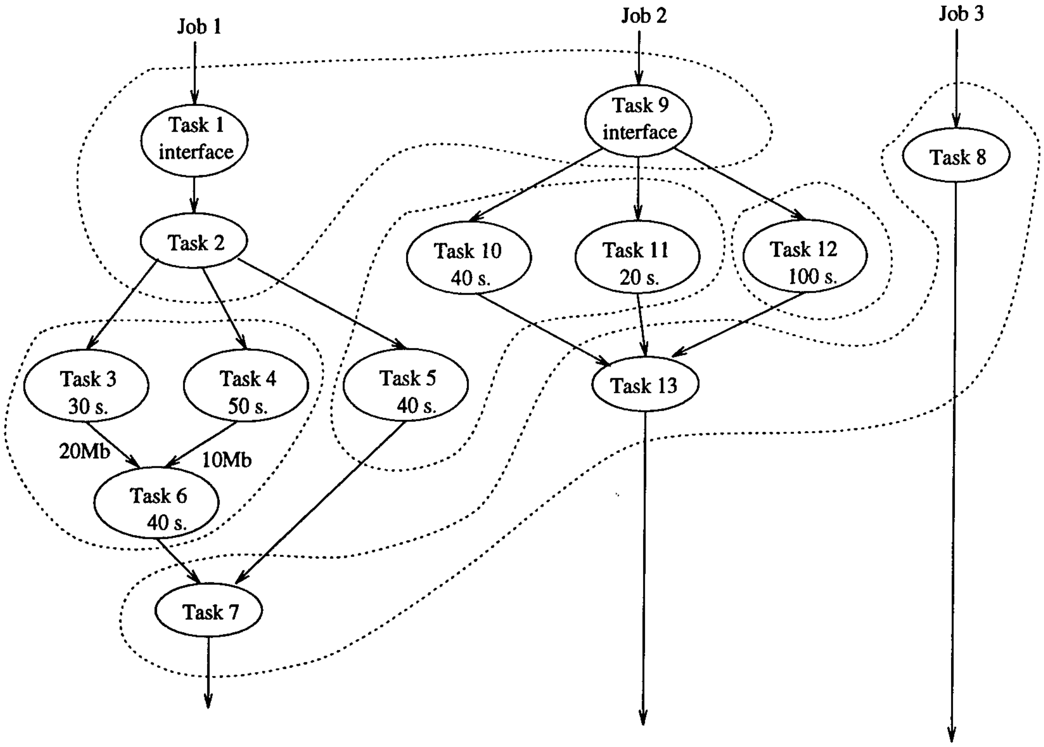

Figure 10.1 Example of task clustering and optimal scheduling.

Figure 10.1 illustrates the processing of three jobs that execute small tasks by a distributed system. It is assumed that users wish these jobs to be completed at about the same time. Generally, the definition of system service quality is difficult to quantified, but can be heuristically qualified in terms of: *quick response*, *short overall completion time*, and *low job completion cost* in network traffic and CPU time. For this example, Task 1 and Task 9 containing interface modules will be better to be grouped in the first cluster and with highest priority to be scheduled for quick response. Tasks 3, 4 and 6 can be considered to form a cluster because there is high network traffic among them (for lower communication cost). For the next group, Task 12 has a longer expected processing time than Task 10 and 11. Thus, the scheduler can either give the Task 12 cluster higher priority (short completion time) or lower the scheduling priority of the cluster with Task 10 and 11. The latter approach will offer the scheduler more flexibility to arrange these two tasks so that they can be processed with lower cost without affecting overall completion time.

Consider a free-market analogy to the optimal scheduling problem. In a common U.S. city, large retail market, like Wal-Mart and Target, and small retail stores, like 7-11 chain stores, coexist (multi-agent). Assume that they are all profitable. It is a common sense that the same product is available in both stores but is less expensive in Wal-Mart than in 7-11. Wal-Mart also has more variety of products to choose from. Why would customers shop in both types of stores? The reasons are obvious, namely:

(1) 7-11 is closer to most customers;

(2) Traffic condition and parking space availability are unknown toward Wal-Mart;

(3) 7-11 has more local goods and neighborhood information, and

(4) Customers do not care about the quality of some products because they are too simple or general.

Timely completion, up-to-date information and *confidence* in estimating construct the major part in decision-making as well as in defining the system service quality. From the discussion, the proposed task clustering and optimal scheduling scheme will contain the following components:

- Multiple mode task scheduler: The example above shows that, in order to acquire performance, the backbone scheduling scheme should be capable of operating in several modes simultaneously. The Osculant scheme has been shown to be flexible and versatile in performing multiple mode scheduling. Jobpost constraints, bidding strategies, and resource management methods can be tailed to accommodate system status and job specifications. In the wake of the development of task organizer technologies, the refinement of task specifications can further improve the system and cost efficiencies.

- Minimum cut algorithms: Our previous studies indicate that clustering or partitioning of tasks are required in order to improve performance because of limitations in communication bandwidth. Based on the amount of data transmission among tasks, systematic partitioning can be implemented by modifying min-cut algorithms [Fiduccia 1982], [Tragoudas 1996]. By reducing overall wire length in the system, min-cut algorithms optimize circuit placement problems. Similar to the task organizer problem, blocks and connections in the system are weighted and directed. Min-cut problems have been shown to be a NP-hard. Therefore, heuristic methods are commonly used in this field.
- Progressive optimization: Osculant has been designed for an on-line scheduling system. Preferably, tasks can be scheduled and executed continually or with a constant waiting time. Thus, a task organizer with progressive output will be ideal. Among many optimization methods, a good candidate is simulated annealing (SA) [Rutenbar 1989]. Simulated annealing method accepts inputs with great temporal and spatial flexibility without seriously affecting results. Furthermore, results form the simulated annealing are retrievable at any moment because of its progressive processing nature. It also means that the organizer waiting time, which is controlled by the cooling schedule in SA, can be used as a control variable in determining the desired level of optimization for the current set of jobs.

11 Conclusions

Improving scheduling performance requires developments and cooperation in several domains. The approaches and directions for the studies are as follows:

- Specification space: Resource specifications must be defined, extracted and arranged so that the system and users can effectively locate and utilize resources. The Osculant Job Profile Generator (JPG) presents the form with which we confront this challenge.
- Self-regulated and flexible scheduling platform: The states of distributed computing systems are difficult to collect. It is even more difficult to synchronize nodes in a heterogeneous system. Osculant jobpost distribution technique presents our approach to balance the state-probing scope and overhead problems. Experimental results show that the system's ethos and performance can be effectively altered and improved by various bidding strategies.
- Efficient resource management: In order to reduce network traffic and system reliability, resources must be properly distributed and replicated in the system. A distributed, self-organized resource management scheme is desired for the target computing environment. The resource forwarding and caching scheme described will be further studied and developed.
- Task organizer: With the arrival of thin-client and component software, high level resources and tasks should be appropriately arranged and scheduled to reduce the operating cost. The task clustering and optimal scheduling explained in Section 10 will be developed in the future.

The anticipated outcomes derive from these research activities are as follows:

- Performance improvement: As stated in the previous sections, achieveing high performance in a distributed computing environment requires cooperation in many fields. Current studies of the Osculant scheduler have developed a prior-execution job profile generator, a self-regulated job announcement process (MMJP) and a very flexible task scheduling scheme with aggressive bidding strategies. With the proposed research topics in the developing task organizer and optimal scheduling techniques, it is anticipated that:

  (1) System load distribution will be further improved. Previous and current load balancing studies focus on distributing system load on a macro (jobs as a whole) and on-demand bases. The proposed studies will explore a new dimension in intra-task scheduling. In other words, the system load will be further distributed in the time axis.

  (2) Job completion cost can be reduced. As mentioned above, the studies of optimal scheduling scheme will refine, determine, or define scheduling parameter of tasks on the run-time. A systematic and self-regulated mechanism will be developed to reduce job completion cost without the intervention of end-users.

- System reliability improvement: The preliminary results of resource management schemes in the Osculant show a major performance enhancement in system throughput and job energy consumption rate. The structure and style of resource management also suggest the high potential of employing Osculant scheme in a cache-based distributed computing system. Future studies will focus on the design of distributed resource management schemes that combines the bottom-up and top-down approaches in order to maintain and to replicate logical resources. It is anticipated that the resulting system will be more reliable by tolerating partial system faults.

Pending future studies include a formal proposal to the NFS to develop a prototype system, and to the Texas Instruments to adapt Osculant for use with their new VLIW-base DSP processors.

Appendix A. Simulation Configuration

In Section 6.5, the simulation results reported were retrieved from the Osculant Simulator with the following configuration:

- 49 nodes in a square mesh topology.
- Node specifications: Node processing power is determined by random using normal distribution with both mean and standard deviation equal to 1. Each node has a communication bandwidth 1 to its neighbors. Bandwidth between two nodes decays at the rate of 5% as the distance (Manhattan distance) increases. That is, Bandwidth = $0.95^{(\text{distance-1})}$. The actual bandwidth between two nodes is affected by network traffic condition which will be discussed later in this section. All nodes have a cache storage with the size of 10 units. Also, resource servers maintain cache validation tables for all resource that they host. The cache validation table records up to 10 different locations.
- Resource types and locations: Locations of resource servers are chosen randomly (uniform distribution). The resources are generated with two parameters: overall size and granularity. A job may need a number (which equals to the granularity value) of resources from servers but the overall size of resources will be constant. In the experiments, there are 6 original resource servers. The average resource granularity of jobs is 3.
- Resource servers take 0.05 time units to perform one cache verification transaction. System maintenance messages (for example, cache validations, jobposts and job assignments) are handled in separate communication channels. Various priority

values are assigned to system messages in order to enforce the correctness and performance of scheduling.

- Task specification: Jobs are generated with two parts using various statistical models: Task processing time is normally distributed with a user specified mean value and a standard deviation of 2. The job generation time interval is 2 time units. This indicates that the system will be saturated with jobs with overall processing time of 98 units in ideal cases.
- In the experiments, job characteristics were changed from CPU intensive to I/O intensive by altering the I/O loads and CPU load of tasks. 2000 tasks are injected into the system with a total simulation time of 5000 units.
- Random and round-robin scheduling are implemented as comparison counterparts. In these two models, there will no jobpost/bidding processes and, therefore, no bidding overheads are introduced.
- Jobpost/bidding constraints: Jobpost constraint is 2 hubs. The deadline for collecting bids in a jobpost/bidding level is 1 time unit.
- Network topology and communication channel: Connection topology determines the number of neighboring nodes and communication channels which nodes can use to access resources. In a point-to-point connection scheme, network load will be added to the source and destination nodes. In a mesh structure, most nodes have 4 neighbors and channels. A through traffic will generally occupy 2 channels but without possessing any I/O ports of nodes on the routing path. Augmented network I/O load to these intermediate nodes will be determined by the number of channels occupied by the traffic and by the amount of transmission traffic passing them.

- Routing: Routes of transmitting resources are determined dynamically in the simulator. It is assumed that packet stream will travel using the shortest path. In a mesh structure, there are multiple shortest paths. The choice of routing path is determined on single transmission-session basis with uniform distribution over all possible shortest paths.
- The experiment results were retrieved using mesh structure and through network traffic contributed to the network load of the intermediate nodes.

Appendix B. Example of Function Result Size Table

```
for CNTRL 1 0
remez GEN 1 3 remez
freqz MAX 0.7 1 reduce2
max SAME 0.2 1
flipud SAME 1 1
filter MAX 1 3 filter
cov SAME 0.2 1
randn GEN 1 1 gen
mean SAME 0.1 1
pow SAME 0.1 1 square2
inv SAME 1 1 square
fft SAME 1 1
mul MAX 0.5 2 mul
or MAX 0.1 1
edge SAME 0.9 1 square
size GEN 0 1
```

Appendix C. Example of Osculant Function Profile

```
add -7.68545e-28 6.69121e-23 -1.12671e-18 1 -1.50834e-12
25 81 289 1089 4225 16641 66049
25 81 289 1089 4225 16641 66049

edge 1.56092e-14 -1.46552e-09 3.20962e-05 31.7973 589.345
25 81 289 1089 4225 16641 66049
1396 3164 9766 35258 135400 533060 2115578

fft -8.97307e-12 1.30193e-06 0.012162 220.605 -505.292
25 81 289 1089 4225 16641 66049
2184 8504 80448 251144 1244288 12350024 271991328

mul 6.20275e-11 -2.0626e-06 0.030444 35.2622 -1037.29
1 36 256 1024 3136 7744 16900
1 432 8192 65536 351232 1362944 4394000
```

Appendix D. Function Result Type Table (FRTT) Example.

Segment of a MATLAB program is shown below:

```
         :                 % IMAGE is an n-by-n matrix
x = mean(IMAGE)'*mean(IMAGE); % An n-by-1 matrix multiplied by
its
                                          transpose form
         :
x = fft(IMAGE);                % Take FFT of matrix IMAGE
         :
x = x + 1;                 % Elements of x are increased by 1
          :
```

Estimate of the first reference for variable *x* is assigned a confidence between 0.1 (mean) and 0.5 (multiplication). The second estimate has a full confidence because of FFT function. The third one has a confidence value of 0.5 because of the operator overload feature in MATLAB. Therefore, the size of variable *x* will be determined by the second reference. The design of FRTT also allows convenient human knowledge intervention. User experiences can be easily adopted into the profile generator.

Appendix E. Structure of the Variable Back Tracing (VBT) Engine

```
function [V, Vsize, Confid]=varbtrac(Variable_Name,VIT_File,User_Command)
  if (is_number) % if a number, return immediately
    return V, Vsize, 1;
  else
    while (not eof(VIT_File))
      READ(VIT_File);
      if (Output_Variable==Variable_Name)
        [Rule,Rconfid,match] = Search_FRST(Function_Type);
        % Rule #1: Input Parameters
        if (Function_Type==INPUT_PARAM) % Rule#1: Input Parameters
          [Output,Input] = Parsing(User_Command);
        [V,Vsize] = mapping(Input);
   % Rule#2: Explicitly defined variables
        elseif (Function_Type==EXP_ASSIGN)
          while (not empty(Input_Variable))
           tok = Parsing(Input_Variable);
           [val,vsize,vline] = varbtrac(tok,VarIFile,User_Command);
            [V,Vsize] = Variable_Dimension_Adjustment(val,vszie);
          end while;
        elseif (RULE==GEN)  % Rule#3: Generating functions
          [V,Vsize] = GEN_Trace(Function_Type,Input_Variable);
        elseif (RULE==CALL)  % Rule#4: Call subroutines
          while (not empty(Input_Variable)) % Trace all inputs
            tok = Parsing(Input_Variable);
            [LV,LVsize] = varbtrac(tok,VIT_FILE,User_Command);
            [Vlist,VsizeList] = append(LV,LVsize);
          end while;
        [V,Vsize] = varbtrac(Vlist,VIT_FILE,User_Command);
        % Rule#5: Size of its relative/predecessor variables
        % Rule#6: Functions that produce the variables
        elseif (match==1)
          while (not empty(Input_Variable))
            tok = Parsing(Input_Variable);
            [LV,LVsize] = varbtrac(tok,VIT_File,User_Command);
            [Vlist,VsizeList] = append(LV,LVsize];
          end while;
          if (VsizeList==1)  % All inputs are scalar, evaluate the expr.
           [V,Vsize] = sceval(Vlist,Original_Program_Expression);
        elseif (RULE==MAX)
           Vsize = max(LVsize);
        end if;
        end if;
      end if;
      [Value,Var_Size,Var_Confid] = append(V,Vsize,Ronfid);
   end while;
   if (sum(Rconfid==1)==1) % only one full confidence estimate
          candidate = index(Rcond==1);
          return Value(candidate), Var_Size(candidate), 1;
   else % Apply Location Information Adjustment (LIA) Method
     [N_Vsize,N_Confid] = LIA(Var_Size,Variable_Location,Var_Confid);
     if (N_Vsize>1) % if the final estimate is matrix
```

```
      return 1, N_Vsize, N_Confid;
    else % if the final estimate is scalar
      return max(Value, 1), 1, N_Confid;
    end if;
  end if;
end if;
```

Appendix F. Main Procedure of Osculant Profile Generator

```
function [Job_STAT]=rc1(samfname,CMAD,DataSize);
  PG_Initialization;
  while (not eof) do
    x = READ(samfname);
    Parsing(x);
    Nest_Conclusion; % Handling loop and branch conclusions
    Function_Classification; % Apply function type decision rules
    Profile_Stage1; % Processing scalar/simple operation types
    case OpType % Process according to structures & functions
       Simple_Matrix: Update_Job_Statistics;
       LOOP:  Push_Loop_Stack(Loop_Parameters);
            Push_BRANCH_Stack(LOOP);
            Update_Job_Statistics;
       BRANCH: if (is_if)
              Initialize_Branch_Load_Array;
              Push_Branch_Stack(IF);
            elseif (is_elseif_else)
              Append_Branch_Load_Array(deltaSTAT);
            else if (is_end)
              BranchType = Pop_Branch_Stack;
              if (BranchType=IF)
                Update_Job_Statistics(max(Branch_Load_Array));
                Update_memory_Statistics;
              else if (BranchType=LOOP)
                delta = Pop_Loop_Stack;
                Update_Job_Statistics(delta*deltaStat);
              end if;
            end if;
       Simple_Scalar: Update_Job_Statistics;
       IO: Update_IO_Statistics;
       Complex_Matrix: Update_Job_Statistics;
       CALL: while (Parsing(Call_List))
            Construct_Pseudo_Instruction_Line;
            % Recursively construct subroutine profile
            [Local_Job_Statistics] = rc1(newsamfname,cmd,DataSize);
            Update_Job_Statistics;
            Update_Memory_Statistics;
            Update_IO_Statistics;
            Update_Branch_Statistics;
          end while;
    end case;
  end while;
  Job_STAT = [Job_Statistics,Memory_Statistics,IO_Statistics,Branch_Statistics];
```

Appendix G Example of the Osculant Simulator User Interface

```
%                      -------------------
%                        M A K E . C F G
%                      -------------------

% This is a sample of config file which gives the full ability to
% set up an approprate enviroment of simulator.

% All the parameters are idenitically in the name domain. The order
% of the inputs does not affect the simulator. BUT, UNFORTUNATELY
% WE ALLOCATE MEMORY AFTER THE VALUE OF 'NODENUM' AND 'JOBNUM'.
% THEREFORE, WE HAVE TO DEFINE 'JOBNUM' AND 'NODENUM' BEFORE GIVE
% ANY OTHER PARAMETERS. IF WE USE THE DEFAULT 'NODENUM' AND 'JOBNUM'
% WE HAVE TO DECLARE 'DEFAULT NUMBER' AT THE BEGINNING.

%        Use the default job number and node number.
%             Default Number

%        The number of jobs that generated from simulator.
            Job Number=
                          jobnum.dat

%        The number of nodes that involved in the simulation.
            Node Number=
                           nodnum.dat

%     Size of cache validation table at resource server
            Cache Table Size=
                            cachetsz.dat

%        Job's resources information. This part of information is
%        quit different from other parts. It requires two parameters
%        The first one is the number of resources, and the second
one
%        is the filename which contains the I/O load according to
the
%        resources index. The default one is that only one resourrce
%        located at node 41 with I/O load of 40.
            Job Resource I/O=
                           resio.dat

%        The number of cash at each node.
            Cash Number At Each Node=
                           cache.dat

%        Simulation time during which all the results are
%        computed.
            Simulation Time=
                           simtim.dat

%        The time out constrain in the posting algorithm.
%        One can set the time constrain to the total
%        simulation time so that simulator will apply
%        repliable posting algorithm in the 23 bits computers.
%        If one set the time up to a number, the posting will
%        not be accepted after that period of delay.
            Posting Time Constrain=
                           poscons.dat
```

```
%    The mutilayer constrain in the posting algorithm.
%    One can set the layer constrain so that no job posting
%    will reach the nodes with a hub distance greater than
%    the layer constrain in one post.
       Posting Layer Constrain=
              layer.dat

%    Location of jobs. We offer a data file name which
%    contains the location (Node index) of where jobs
%    generated.
%    The file is the location of the the where jobs
%    generated according to the index order of jobs.
%    File data are store in ASCII form.
       Job Location=
              jobloc.dat

%    Job's generation time. Everything is defined above.
       Job Generated Time=
              jobgen.dat

%    Job's computation time.
       Job Computation Load=
              jobcpu.dat

%    Job's memory requirement.
       Job Memory Requirement=
              jobmem.dat

%    Job's I/O location which we give the main resources
%    index at location where the job's data is located on.
       Job I/O Location=
              resloc.dat

%    Job's profile size.
        Profile Size=
              jobprf.dat

%    Node's CPU speed.
       CPU Speed=
              cpu.dat

%    Node's memory capacity.
       Memory=
              mem.dat

%    Node's I/O speed (Bandwidth).
       I/O Speed=
              io.dat

%    Node's local load generator. This file is a
%    little different from others. It offers two
%    parameters of pdf of Gaussian.
%       Local Load=
%              loclod.dat

%    In this file, we supply a masking a logical
%    link matrix. '1' and '0' presents the connection
%    between any two nodes. The connection matrix is
%    a NodeNum by NodeNum matrix.
       Network communication Link Matrix=
                              loglin.dat

%       Physical links between any two nodes which is
%       defined above.
            Physical Link Matrix=
                              phylin.dat

%    Bidding strategies
%        1: Bid_On_Completion_Time
%        2: Bid_On_Utilization_&_Completion_Time
%        3: Random bid
%        4. Round-robin scheduling
%        5. Multiple bid strategy: Dynamic job profile model
%    Future extention: Nodes can apply different bidding strategies
and
%             the strategies can vary in time.
     Bidding Strategy=
                 bidstrgy.dat

     Job Predict Error=
                 jobperr.dat

            Simulated Annealing Auction=
                              simannau.dat

            Node Battery Life Time=
                              batltime.dat
%    Resource Distribution Managament Method
%        1: Plain Method
%        2: Request Frequency Method
            Resource Distribution Management=
                              resdistm.dat

%    Network Load Model
%        0: Point-to-point Model
%        1: Ethernet Model
     Network Load Model=
                 netloadm.dat
```

Appendix H. Osculant Supported Ph.D. Dissertation

Author: Ansari, Ahmad Reza

Title: "*Characterization of Multicomputer Interconnection Network Performance Under Real-Time and Non-Real-Time Traffic*"

Ph.D. Thesis, University of Florida, 1995

Please refer to the attachment.

Appendix I. References

[Blake 1991] B.A. Blake, K. Schwan, "Experimental Evaluation of a Real-Time Scheduler for a Multiprocessor System", IEEE Tran. Software Engineering, vol. 17, no. 1, Jan. 1991.pp. 34-44.

[Chow 1996] R.Y. Chow and T. Johnson, Distributed Operating Systems and Algorithms, Addison-Wesley, 1996.

[ComponentWare 1997] "ComponentWare: Component Software for the Enterprise", I-Kinetics Inc., April 1997.

[Ferguson 1988] D. Ferguson, Y. Yemini, and C. Nikolaou, "Microeconomic Algorithms for Load Balancing in Distributed Computer Systems", IEEE 1988, pp. 491-499.

[Fiduccia 1982] C.M. Fiduccia and R.M. Mattheyese, "A Linear-Time Heuristic for Improving Network Partitions", IEEE 19th Design Automation Conference, 1982, pp. 175-181.

[Gagliano 1995] R.A. Gagliano, M. Fraser and M. Schaefer, "Auction, Allocation of Computing Resources", Comm. of ACM, vol. 38, no. 6, June 1995.

[Goscinski 1991] A. Goscinski, Distributed Operating System The Logical Design, Addison-Wesley, 1991.

[Goulde 1997] M.A. Goulde, "Searching for the Networked Computer", Patricia Seybold Group, May 1997.

[Klingerman 1986] E. Klingerman and A.D. Stoyenko, "Real-Time Euclid: A language For Reliable Real-Time Systems", IEEE Tran. on Software Engineering, Vol. 12, No. 9, Sep. 1986, pp. 941-949.

[Liu 1973] C.L. Liu and J.W. Layland, "Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment", J. of ACM, 20, Jan. 1973, 46-61.

[MATLAB 1992] MATLAB Reference Guide, The Mathworks Inc., 1992.

[MATLAB 1993] MATLAB External Interface Guide, The Mathworks Inc., 1993.

[Ni 1985] L.M. Ni, C.W. Xu, and T.B. Gendreau, "A Distributed Drafting Algorithm For Load Balancing", IEEE Trans. on Software Engineering, Vol. 11, No. 10, Oct. 1985, 1153-1161.

[Park 1989] C.Y. Park and A.C. Shaw, "Experiments with a Program Timing Tool Based on Source-Level Timing Schema", IEEE Computer, May 1991, 48-57.

[Park 1993] C.Y. Park, "Predicting Program Execution Times by Analyzing Static and Dynamic Program Paths", J. of Real-Time Systems, May 1993, 31-62.

[Puschiner 1989] P. Puschiner and C.H. Koza, "Calculating The Maximum Execution Time of Real-Time Programs", J. of Real-Time Systems, 1, 159-176(1989).

[Ramamritham K. Ramamritham, J.A. Stankovic, and W. Zhao, "Distributed Scheduling of Task

1989] with Deadlines and Resource Requirements", IEEE. Tran. on Computers, Vol. 38, No. 8, August 1989, pp. 1110-1123.

[Ramamritham 1994] K. Ramamritham, and J.A. Stankovic, "Scheduling Algorithms and Operating Systems Support for Real-Time Systems", Proceeding of IEEE, Vol. 82, no. 1, Jan. 1994, pp. 55-67.

[Rutenbar 1989] R.A. Rutenbar, "Simulated Annealing Algorithms: An Overview", IEEE Circuit And Device Magazine, January 1989.

[Schaller 1997] R.R. Schaller, "Moore Law: Past, Present, and Future", IEEE Spectrum, June 1997.

[Shaw 1989] A.C. Shaw, "Reasoning About Time in Higher-Level Language Software", IEEE Tran. on Software Engineering, Vol. 15, No. 7, July 1989, 875-889.

[Shin 1988] K.G. Shin and Y. Chang, "Load sharing in distributed real-time systems with state change broadcasts", IEEE Trans. on Computer, Vol. 38, No. 8, Aug. 1988, 1124-1142.

[Shin 1995] K.G. Shin and Y. Chang, "A Coordinated Location Policy for Load Sharing in Hypercube-Connected Multicomputers", IEEE Trans. on Computers, Vol. 44, No. 5, May 1995, 669-682.

[Smith 1980] R.G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver", IEEE Trans. on Computers, Vol. 29, No. 12, Dec. 1980, 1104-1113.

[Thamas 1997] W. Thomas, "Castanet: A New Way to Deliver Software and Content", Netscape Communication Corp., 1997.

[Tragoudas 1996] S. Tragoudas, "Min-Cut Partitioning on Underlying Tree and Graph Structure", IEEE Trans. on Computers, Vol. 45, No. 4, April 1996, pp. 470-474.

[Wu 1997] H. Wu, C. Chen and F.J. Taylor, "Osculant: A Multiprocessor Self-Organizing Task Scheduler", Proceeding of IEEE Performance, Computing and Communication Conference, Phoenix AZ, Feb. 1997, pp. 35-41.

[Wu 1997] H. Wu and F.J. Taylor, "Osculant: A Self-Organizing Scheduling Scheme in a Network Computing Environment", HSDAL, Department of Electrical and Computer Engineering, University of Florida, Dec. 1997. Submitted for review at IEEE Trans. on Parallel and Distributed Systems, Jan. 1998.

[Wu 1997] H. Wu and F.J. Taylor, "Osculant Job Profile Generator: Extracting Job Profile in Source Code Level", HSDAL, Department of Electrical and Computer Engineering, University of Florida, Aug. 1997, Submitted for review at IEEE Trans. on Software Engineering, Sep. 1997.

Acknowledgments

Osculant is being developed under DOD sponsorship [NAVY 00039-44-C-0163, http://www.hsdal.ufl.edu] with a major contribution from Hewlett Packard.

ATTACHMENT

CHARACTERIZATION OF MULTICOMPUTER INTERCONNECTION NETWORK PERFORMANCE UNDER REAL-TIME AND NON-REAL-TIME TRAFFIC

By

AHMAD REZA ANSARI

A DISSERTATION PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

UNIVERSITY OF FLORIDA

1995

TABLE OF CONTENTS

CHAPTERS

APPENDICES

LIST OF FIGURES

Abstract of Dissertation Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Doctor of Philosophy

CHARACTERIZATION OF MULTICOMPUTER INTERCONNECTION NETWORK PERFORMANCE UNDER REAL-TIME AND NON-REAL-TIME TRAFFIC

By

Ahmad Reza Ansari

August 1995

Chairman: Dr. Fred J. Taylor
Major Department: Electrical Engineering

Future multicomputer networks have to satisfy the diverse performance requirements of the parallel real-time and multimedia applications. These applications usually generate multiple traffic classes which demand different performance requirements. This mixture of loads typically consists of a guaranteed class whose packets require bounds on latency or throughput and a best-effort class which requires good average performance.

In this dissertation we start off by examining closely the assumptions and requirements of multicomputer network design and reevaluate their parameters to see how they could deliver the high performance required by these diverse applications.

We analytically model the latency in k-ary n-cubes, generalized hypercubes, and WK-Recursive networks, under cut-through and store-and-forward switching schemes, with or without contention. The network analysis under no contention presents the base network latency and allows us to study the effect of various types of wire and switch delays on the network performance. We develop closed form expressions for latency and its variance under contention in buffered direct networks. The contention models are merged with the base network results to obtain the complete latency models for the multicomputer networks.

To verify the validity of the models a network simulator is developed. This simulator allows evaluating the interconnection topology, interprocessor routing, and communication flow control. The data collected from the simulator were used to test the developed models and also served as the primary source whenever it was difficult to derive accurate analytical models.

Finally, the dissertation establishes a paradigm for the efficient and reliable mixing of guaranteed and best-effort traffic in multicomputer networks. Unlike the previous work in this area, we propose architectural features which exercise efficient, fine-grain control over the interaction of packets. In order to optimize for the performance requirements of each class, the architecture employs different routing and switching strategies to manage the two traffic classes. We bound the intrusion of each traffic class on the other by low-level bandwidth allocation. The software or the higher level hardware can utilize these bounds to provide the quality of service required by the different application.

CHAPTER 1
INTRODUCTION

1.1 Motivation

With the advent of parallel real-time and multimedia applications which demand high levels of performance and dependability, the design requirements of supercomputers are being rewritten. Massively parallel machines of the future will not merely be engaged in highly computation-intensive scientific applications. Recent real-time applications, such as on-demand multimedia services and interactive television, require a level of performance which is produced only by large concurrent computers. The predictability required by real-time applications makes it essential for the systems to possess efficient and predictable interprocessor communication networks which are highly fault-tolerant.

The network may connect the processing nodes of a message-passing multicomputer [6], or the processors and memories of a shared-memory multiprocessor [9]. In either case, the performance of the parallel computer depends heavily on the network latency and its throughput. Furthermore, the network accounts for a large fraction of the cost and power dissipation of the machine.

The focus of this research is on the message-passing concurrent computers, also known as multicomputers, such as the nCUBE [37], which consist of many computing nodes that interact with one another by sending and receiving messages over communication channels between the nodes . The nodes in a multicomputer can be laid out in space using different topologies which possess different routing and fault-tolerance characteristics. A good interconnection structure in general should have a low number of links per node. a small internode distance. and a large number of alternate paths between a pair of nodes for fault-tolerance.

Although multicomputer network design has traditionally emphasized providing low-latency communication. modern parallel applications require additional services from the interconnection network [10]. Multimedia and real-time applications. such as scientific visualization and process control. necessitate control over delay variance and throughput. in addition to low average latency [22]. These applications usually generate two distinct classes: guaranteed and best-effort traffic. which possess different communication requirements. Guaranteed traffic. such as control or audio/video. may necessitate explicit performance guarantees and mandate deterministic or probabilistic bounds on throughput or end-to-end delay. while best-effort traffic. such as data transfer. may tolerate more variability in delay. at the cost of improved average latency.

The performance of a multicomputer network is directly affected by the choice of the routing and switching schemes. The majority of the contemporary multicomputers employ oblivious routing schemes which guarantee deadlock freedom [38]. However, since oblivious routing policies prevent full utilization of the network. in large multicomputers, these routing algorithms are not able to provide the desired network performance. In these machines, the average message traffic, which is at least a linear function of the total number of nodes, grows faster than the bisection area of the network which, due to the 3D construction of the machine, grows only as $N^{2/3}$, where N is the number of nodes. This creates local congestion at certain parts of the network [11].

To improve the network performance of these highly parallel machines. the routing mechanism has to be able to diffuse the local congestions by adaptively utilizing the available resources in the network. In contrast with the oblivious routing in which the message trajectories are unique. in an adaptive routing scheme. they are continuously perturbed based on the condition of the network. However. this adaptivity contradicts the predictability requirement which is essential for a real-time system.

On the other hand. most modern multicomputer networks try to reduce the average communication latency by implementing certain switching techniques which avoid the unnecessary delay at the intermediate nodes. These low-latency techniques often impinge on control over packet scheduling which further complicates the effort to provide predictable or guaranteed service. In particular. wormhole and cut-through switching [30] decentralize bandwidth allocation and packet scheduling by allowing an incoming packet to proceed directly to the next node in its route if a suitable outgoing link is available.

Handling a mixture of disparate traffic classes affects the suitability of architectural features in multicomputer routers. While the router alone cannot satisfy application performance requirements, design decisions should not preclude the system from providing necessary guarantees. Servicing guaranteed traffic requires control over network access time and bandwidth allocation. The router should bound the influence best-effort packets have on these parameters. The software or higher level hardware can utilize these bounds to provide the quality-of-service requirements through packet scheduling and resource allocation for communicating tasks. Additionally, the design should not unnecessarily penalize the performance of best-effort packets by allocating the entire resources to the guaranteed traffic.

1.2 Interconnection Networks

Concurrent computers can be grouped into two major categories, shared-memory multiprocessors and message-passing multicomputers. In shared-memory multiprocessors a single memory address space is shared by all the processors and they communicate with one another through this shared space. To provide an equal distance between any processor and any memory, most contemporary multiprocessors have adopted multistage interconnection networks which display this equidistance property [25, 39]. In multistage networks, which are also referred to as indirect networks, the communication between any pairs of nodes occurs through multiple stages of the network. In contrast, message-passing multicomputers do not provide a global address space and their memory is distributed across the processing nodes. In multicomputers, processors can only access their local memory directly and access to remote memories is through messages sent to the node which contains the memory. Today's multicomputers predominantly use direct networks represented by grids and meshes [17, 42].

Although the topological equidistance of the indirect networks makes them suitable for shared memory multiprocessors, in general, these networks are not very attractive for massively parallel systems (10^3 to 10^6 processors). As the number of processors in these networks increases, the distance between any two nodes will increase which diminishes the performance benefits of scaling. On the other hand, the inherent locality of the direct networks, unequal distance between nodes, can be exploited to make the system scalable to large numbers of processors. Due to their scaleability, direct networks have gained more acceptance in multiprocessors which was once dominated by indirect networks. Examples of multiprocessors using direct networks are Tera Computer's TERA machine [2], and MIT's Alewife.

Different schemes have been employed to improve the latency and throughput of multicomputer networks. One approach is to reduce the frequency of communication through the network by using local caches for each processor [33]. Another method is to hide the latency by overlapping the communication time with useful work. Different approaches to this method have been used by several groups. The MIT J-machine uses context switching [17] and DASH [33] employ multithreading to complement memory access due to cash miss.

While caches and multithreading improve some of the problems associated with nonideal networks, they introduce new problems. Using caches requires cache coherency techniques which are complex and costly. Furthermore, despite the fact that using caches reduces the amount of data traffic in the network, the cache coherency protocol introduces new traffic which increases the overall network communication. Also, some applications such as FFT and matrix transpose display poor communication locality which limits the benefits of having caches. Multithreading also involves the overhead of context switching and is limited by the amount of parallelism available in applications. Finally, the uncertainty of cache hit/miss causes unpredictable memory access delay, which in turn makes the system less predictable and consequently less desirable for real-time applications.

Research in Local or Wide Area Networks (LAN/WAN) considers techniques for the effective mixing of multiple traffic classes in a communication fabric [7, 4]. However, the design trade-offs for parallel machines differ significantly from those in a heterogeneous, distributed environment. All multicomputers, and fine-grain machines in particular, possess rather limited hardware resource per node which limits the amount of internal message buffer that each node can devote to routing. In these machines, router design trade-offs reflect the large network size and the tight coupling between nodes. Speed and area constraints motivate single-chip solutions, including designs that integrate the processing core and the communication subsystem [17]. Also, these networks are usually very tightly coupled physically. This creates a low signal propagation delay across their networks which necessitates the use of hardware, rather than software, support for message transport, routing, and buffer management. The hardware mechanisms allow simple and efficient cycle-by-cycle flow control schemes. On the other hand, this fast channel speed requires fast routing circuitry which in turn limits the amount of information that the routing algorithm can afford. In much the same way, the multicomputer networks, unlike geographically distributed networks, are usually installed in well-protected environments. Hence, the signal transmission error rate across a channel is extremely small and generally negligible. Finally, multicomputer networks almost always employ very regular network topologies for their connections which allows one to define simple algorithmic routing procedures that eliminate the requirements to store routing tables.

1.3 Real-time Applications and their Communication Requirements

In a real-time system the correctness of an operation depends not only on its logical correctness, but also on its timeliness. A real-time application is usually comprised of a group of cooperating tasks which are invoked in a *periodic* or *aperiodic* manner. Each task must finish its execution within a specified time, called deadline. Two important characteristics of real-time systems are *predictability* and *reliability*. The definition of predictability may vary among different tasks. *Hard* real-time tasks require a 100% guarantee that their constraints will be satisfied. Some other tasks require *probabilistic* or *run-time deterministic* guarantees [43]. To satisfy any kind of deadline guarantee, the complete characteristics of the tasks such as their execution and arrival times must be known *a priori*. In practice, it is very difficult to know the exact values of these parameters and usually the worst-case values which are derived through simulation and testing are used. For a real-time system to operate properly, all aspects of the system such as the architecture of the node, the communication subsystem, the operating system, and the programming languages have to support the notion of deadline guarantee at every level of abstraction.

The architecture of real-time computers can be studied at two different levels: the *node* level and the *system* level. At the node level, the system has to provide predictability in instruction execution, interrupt handling and communication with the outside of the node. For example, using virtual memory or cache degrades the predictability of the node [43]. At the system level, predictability is achieved by studying internode communication and fault-tolerance.

In the earlier real-time systems, the interconnection network was usually based on a broadcast bus which, due to its inherent bottle-neck, cannot deliver the performance and reliability required by recent real-time applications. Point-to-point interconnections are prime candidates for these applications due to their intrinsic fault-tolerance and high bandwidth. However, it is more difficult to achieve predictability in multihop point-point networks. This is mainly due to the fact that in multihop networks the characteristics of the various traffic streams can change as they pass through the network. For example, if the input to the system is generated according to a Poisson process, while it traverses through the network it may not generate inputs with the same distribution on the intermediate nodes. This causes even more problems when both real-time and nonreal-time traffic exist in the network. Although the traffic patterns generated by nonreal-time applications normally follow a Poisson distribution, real-time traffic such as voice or video is usually bursty.

The influence of the two traffics on one another causes the nonreal-time traffic to become bursty, as well. Although in nonreal-time communication, the focus of the communication protocol is on the aggregate, rather than on each individual packet, and the mean message delivery is the parameter of interest, the dependence on the real-time traffic makes the analysis of the nonreal-time traffic more difficult.

The performance requirements of real-time communications are usually expressed by the clients in terms of the delay, throughput, or reliability. These requirements are normally specified in terms of deterministic or statistical bounds. Deterministic bounds can be viewed as statistical bounds that are satisfied with probability one [22]. The delay requirements can be specified as bounds on the delay or bounds on the delay jitter. Throughout our study, we use average latency to determine the performance of nonreal-time communication and latency standard deviation or coefficient of variation, to evaluate the predictability of real-time communication.

1.4 Dissertation Outline and Summary

In this dissertation, we investigate the effect of architectural and load parameters on the communication of traffic in multicomputer networks. We demonstrate, through analytical modeling and simulation, how different network architectures can accommodate different performance requirements. We also present a new paradigm for communication of two general classes of traffic in multicomputer networks. These classes consist of a time-sensitive guaranteed and time-insensitive best-effort class. The remainder of this dissertation is organized as follows.

Chapter 2 serves as the background and presents the material which will act as a foundation for the dissertation. It will describe different variables influencing the communication in a multicomputer network and reviews the previous work in the area. The chapter describes the network topology focusing on k-ary n-cube, Generalized HyperCube (GHC), and WK-Recursive structures. The optimal topology depends critically on a variety of design constraints. The network topologies suggested in the literature are reviewed. The chapter also discusses various communication techniques developed for multicomputer networks including switching techniques, routing functions and virtual channel flow control.

Chapter 3 presents a simulation model for studying the impact of routing and switching on interconnection network performance. This simulator allows evaluating the interconnection topology, interprocessor routing, and communication flow control. The data collected from the simulator are used to test the developed models and also serves as the primary source whenever it was difficult to derive accurate analytical models. Using the simulator model, we investigate how switching schemes affect the network's ability to service multiple traffic classes.

In Chapter 4, through analytical modeling and simulations, we examine closely the assumptions and requirements of multicomputer network design and reevaluate their parameters to see how they could achieve the high performance requirements. We model the latency in k-ary n-cube, generalized hypercubes, and WK-Recursive networks under cut-through and store-and-forward switching schemes with or without contention. The network analysis under no contention presents the base network latency and allows us to study the effect of various types of wire and switch delays on the network performance. We develop closed form expressions for latency and its variance under contention in buffered direct networks. The contention models are merged with the base network results to obtain the complete latency models for the multicomputer networks.

In Chapter 5, we evaluate the ability of wormhole, virtual cut-through, and store-and-forward switching to accommodate different performance requirements. We investigate, based on simulation results, how each switching scheme can affect the performance and the predictability of a single class of traffic.

In Chapter 6, we establish a paradigm for the efficient and reliable mixing of guaranteed and best-effort traffic in message-passing multiprocessors. We propose architectural features which exercise efficient fine-grain control over the interaction of packets. To optimize for the performance requirements of each class, the architecture employs different routing and switching strategies to manage the two traffic classes. We provide tight bounds on the intrusion of best-effort traffic on guaranteed packets by the low-level control of the network access time and bandwidth allocation.

In Chapter 7, we conclude the dissertation with a summary of the research results presented.

CHAPTER 2
BACKGROUND

The interconnection network is an essential part of a multicomputer system which directly affects its performance, reliability, and programmability. Due to the technological advancements in processor design, the computing power of individual processors has improved substantially and there is more need than ever to provide a communication network which does not become the bottleneck in the multicomputer system. Also, to provide the astronomical computation power requirements of some applications, the number of nodes in these systems have to increase which elevates the requirements for failure resiliency in the systems. On the other hand, the quality of services (QOS) imposed on the network by different applications are completely distinct and the network has to satisfy these services efficiently. Finally, the efficiency of the interconnection network determines the granularity level of the system and directly dictates to the programmer how to structure his code to utilize the system maximally.

This chapter presents background material which will act as a foundation for the remaining chapters. We will describe different variables influencing the communication in a multicomputer network and review the previous work in the area.

2.1 Topology

In this section, we will analyze multicomputer networks from a topological point-of-view and examine how certain design decisions affect the performance and reliability of the entire network. We will analyze three groups of interconnection networks. generalized hypercubes. k-ary n-cubes. and WK-Recursive structures.

2.1.1 Terminology

One of the most natural and widely used models to represent a multicomputer network is through a strongly connected. directed graph. $M = G(N.C)$. The vertices of M are a set of nodes. N. which physically correspond to the multicomputer nodes each containing a computation unit. a communication unit and local memory. The edges are a set of uni-directional links or channels. $C \subseteq N \times N$ which represent the physical connectivity of the network. A channel $(n_1.n_2)$ is bi-directional if $(n_1.n_2) \in C \Rightarrow (n_2.n_1) \in C$. Interconnection topologies are evaluated in terms of the following metrics:

Symmetry: A network is symmetric if there exists a homomorphism which maps any node in the network onto any other node [40]. All the nodes in a symmetric network have identical view of the rest of the network. A ring and a tree are examples of symmetric and asymmetric networks. respectively. Symmetric networks simplify many resource management problems such as load balancing. On the other hand, asymmetric networks are shown to be ill-suited for general purpose multicomputer networks [41, 48]. The inherent topological bottlenecks in asymmetric networks usually limit the interprocessor communication in the network. For example. the root of a tree is much more subject to saturation than any other nodes and it becomes a bottleneck. The only exception to this case. are special purpose architectures in which the pattern of internode communication matches the network topology.

Network Connectivity and Bisection Width: The efficiency and fault-tolerance of a network is directly a function of its connectivity. By definition. connectivity is the minimum number of nodes or links which must fail to partition the network into two or more disconnected subnets.

If a system possesses high link or node connectivity. it is more resilient to failures. Of course. this is true as long as the network provides some mechanism. such as adaptive routing, to take advantage of the extra connections. Furthermore. greater connectivity improves performance by reducing the paths from a source to a destination.

The Bisection Width or the Channel Bisection. η. of a network is the minimum number of channels that has to be cut to partition the network into two equal parts. Bisection width determines the rate at which communication can take place between different halves of a computer (bisection bandwidth). A low bisection bandwidth is an indicator that bottlenecks may arise in some section of a network [47].

Network Degree: The number of links incident on a node is referred to as the degree of the network and is represented by d. The network degree directly determines the number of pins on each node which is limited by the technology. This constraint affects the maximum connectivity of the networks and also restrains the maximum data rate into and out of a network.

Network Diameter: In a network of n nodes. the diameter is defined as $D = max\{d_{ij} \mid 1 \leq i.j \leq N\}$, where d_{ij} is the distance between nodes i and j along the shortest path. Diameter of a network is used by many as the Figure Of Merit (FOM) for the network. This is one reason for popularity of *dense* networks such a binary n-cubes which possess large number of nodes and relatively low diameters.

There is usually a trade-off between the node degree and the diameter of a network. A structure with a low degree has a large diameter and one that has a low diameter usually possesses a large node degree. The completely-connected and single loop structures represent the two extremes. The fully connected topology with n nodes has unit diameter but $O(n^2)$ links: however. a ring structure has $O(n)$ links and $O(n)$ diameter. The *cost* function defined as the product of the diameter and the node degree ($\xi = D \times d$) is therefore a good criterion to measure the performance of a structure.

2.1.2 VLSI Constraints

In a general-purpose multicomputer. every node communicates with all the other nodes by sending messages through the network. Ideally. the more connections the network possesses. the more efficient the communication will be. However. due to the constraints imposed by the technology. a highly connected network. except for small number of nodes, would be impractical.

As the number of connections increases. the node degree and the channel bisection. η. of the network will increase. In practice. these two parameters are restricted by the node size and the wire bisection bandwidth. respectively. A node with degree d and channel width W. requires Wd connections. Practically. there is a limit to the number of pins on a chip or connections on a board. Also. a direct network is constrained by the cost of its wire bisection. A network's wire bisection width, η_w, is defined as the minimum number of wires to be cut to divide the network into two equal parts. The wire bisection width is limited by wiring density and the total system size, each of which is determined by layout technology, system cost. and power dissipation. respectively. Hence, the wire bisection, η_w, is a good measure of the cost of the network and should be held constant when comparing networks.

In our analyses, we assume that these two parameters are bounded and change the other parameters of the network, such as the channel width, to find the optimal configuration. This is in contrast with the traditional analysis of networks under constant channel bandwidth which favors networks with high dimensionality, such as the binary n-cube over low dimensional networks such as tori. The constant bandwidth assumption is not consistent with the properties of VLSI technology. Networks with high number of dimensions require more and longer wires and more pins which make them cost more and run more slowly than low-dimensional networks.

The Wire Length is another parameter which is important in the evaluation of a multicomputer network. The wire length in a network puts an upper bound on the speed and power dissipation of the network. If wires are sufficiently short, their propagation delay is usually modeled as logarithmically dependent on the wire length. On the other hand, for long wires the delay will be limited by the speed of light and is normally assumed a linear function of the channel length [13]. In other words, if the wire length is ℓ,

$$T_{prop} \propto \begin{cases} 1 + \log \ell & \text{for small } \ell \\ \ell & \text{for large } \ell \end{cases} \tag{2.1}$$

We will assume constant, linear, and logarithmic models for the wire delay, when we investigate the base netw.rk latency in chapter 4.

2.1.3 Generalized Hypercubes

A Generalized HyperCube (GHC) is a structure with an arbitrary number of nodes which is obtained by a complete generalization of the hypercube networks, allowing them to have different number of nodes in each dimension. GHCs are more cost-effective than regular hypercubes and possess very good fault-tolerance [3].

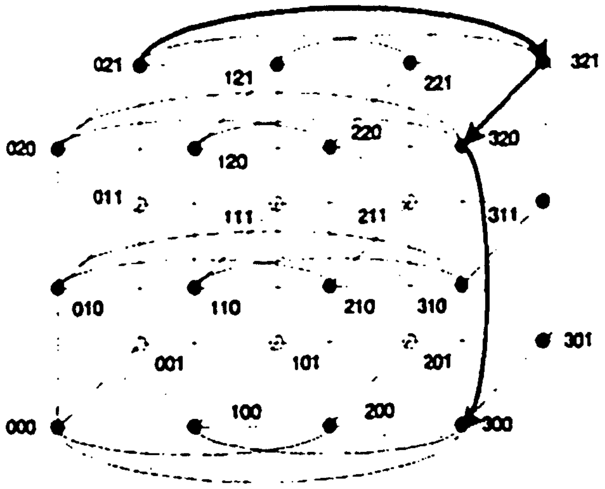

Figure 2.1: A 4 × 3 × 2 GHC structure.

Mixed Radix Representation

If an n-dimensional GHC possesses k_i nodes in its i-th dimension. the total number of nodes in the GHC will be.

$$N = k_n \times k_{n-1} \times \ldots \times k_1$$

Each processor X between 0 and $N - 1$ is expressed as an n-tuple $(x_n x_{n-1} \ldots x_1)$ for $0 \leq x_i \leq (k_i - 1)$. Associated with each x_i is a weight w_i, such that $\sum_{i=1}^{n} x_i.w_i = X$. where $w_i = \prod_{j=1}^{i-1} k_j = k_{i-1} \times k_{i-1} \times \ldots \times k_1$ for all $1 \leq i \leq n$.

Description of GHC Structure

Each processor $X = (x_n x_{n-1} \ldots x_{i+1} x_i x_{i-1} \ldots x_1)$ will be connected to processors $(x_n x_{n-1} \ldots x_{i+1} x'_i x_{i-1} \ldots x_1)$ for all $1 \leq i \leq n$ where x'_i takes all integer values between 0 to $(k_i - 1)$ except x_i itself. A $4 \times 3 \times 2$ GHC is shown in Figure 2.1. For the sake of clarity, the connections in this figure are not shown for the nodes represented by white circles. Figure 2.2-a also depicts a 4 × 4 GHC.

The GHC structure consists of n dimensions with k_i number of nodes in the i-th dimension. A node in a particular axis is connected to all other nodes in the same axis. Therefore, from any node there are $(k_i - 1)$ links in the i-th direction, hence degree of a node $d = \sum_{i=1}^{n}(k_i - 1)$. Each link is connected to two processors, therefore the total number of links in GHC structure is $(N/2)\sum_{i=1}^{n}(k_i - 1)$. Hamming distance between two nodes differing in their addresses only in the i-th coordinate is unity, and the Hamming distance between any two nodes is the sum of the number of coordinates in which the addresses differ. The addresses can differ at maximum n coordinates. Thus, the diameter of the structure, $D = n$.

There are d (d = degree of a node) alternate paths between any two nodes of the GHC. For less than d faults in the system, the worst case distance between two connected nodes is $n + 1$. There are h disjoint paths of equal length h between any two nodes separated by the Hamming distance h.

Deterministic Routing in Generalized Hypercubes

To route messages in GHCs, at each node, the destination address is compared to the node address. If the addresses match, the node accepts the message. If they do not, the node transmits the message along the direction of the first differing digit. The process continues until the destination is reached. As the message gets closer to the destination, it moves into subcubes of successively smaller dimension in which the destination node resides. Using the above scheme, it is obvious that the path from node i to j is not commutative. In Figure 2.1, a message is routed from node 021 to node 300 in a 4 × 3 × 2 GHC.

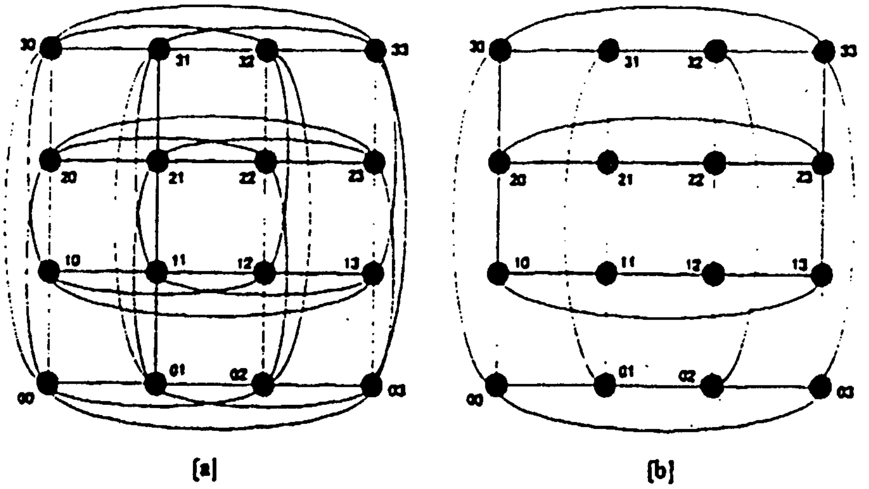

Figure 2.2: Different hypercube networks. (a) GHC with $n = 2$ and $k = 4$: (b) Binary 2-cube.

2.1.4 k-ary n-cubes

A k-ary n-cube has $N = k^n$ nodes. We refer to n as the dimension and to k as the radix of the cube. Each node is connected to $2n$ neighbor nodes and each dimension contains k nodes linearly connected. A node in the k-ary n-cube can be identified by n-digit radix k address. $a_0,\ldots.a_{n-1}$. The i-th digit of the address. a_i. represents the node position in the i-th position. Figure 2.2-b shows a binary 2-cube.

An interesting issue regarding k-ary n-cube networks is how to choose k and n for given N nodes to achieve the best performance. Without considering any implementation constraints, high-dimensional networks appear to perform better because of their lower diameter. A smaller diameter implies reduced latency and. more importantly, much less channel contention.

However, the optimal choice of k and n critically depends on a variety of design constraints. For k-ary n-cube networks with wraparounds (torus), the bisection width is

$$\eta = 2k^{n-1} = \frac{2N}{k} = \frac{2N}{\sqrt[n]{N}} \tag{2.2}$$

If the wraparound connections do not exist (torus). η is halved. For a fixed-size network (fixed N). as network dimension n grows. η increases superlinearly. Since the width of each channel is derived as $W = \frac{2w}{\eta}$. the channel width. W. decreases rapidly as the dimension n grows. For a given message length. narrow channels increase message latency. overwhelming the advantages of high-dimensional networks. In a comparative study based on normalized channel width on the assumption of constant wire bisection. Dally [13] showed that networks with two or three dimensions provide better performance than high-dimensional networks. In addition. low-dimensional networks are preferred because wire lengths increase with network dimension. The significance of increased wire length is discussed extensively by Aggarwal [1].

Wire bisection is not the only constraint that applies to network implementation. In practical systems. channel width may be constrained by node sizes rather than wire bisection [14]. Under the node size constraint. moderate (3, 4 or 5)-dimensional networks are more attractive. Since the number of pins is limited by the node size, channel width decreases as the network dimension increases. However. when compared to the constant wire bisection limitation. node size limitation decreases channel widths much more slowly. As the network dimension increases. the advantages of high-dimensional networks overwhelm the reduced channel width. Consequently, under pin limitation, two-dimensional networks give much worse performance than do networks of moderate dimensions especially under heavy loads. An analytical comparison under the pin limitation was done by Agarwal [1]. He also showed that the optimal dimension is highly sensitive to system parameters such as packet length.

There are two alternative ways to implement k-ary n-cube networks. with (torus) or without (mesh) wraparound channels. Torus networks are symmetric in the sense that the network topology is identical when viewed from any node. The symmetry allows even utilization of network resources. The wraparound paths of torus networks also provide a smaller network diameter. reducing channel congestion as well as average distance. Using bidirectional channels. the average distance in the torus network is $n\frac{k}{4}$, which is much shorter than the mesh network's distance. $n\frac{k}{3}$. for high radix (k). On the other hand. torus networks require more complicated routers. Since the wraparound channels introduce an additional possibility of deadlock. more resources are needed to prevent deadlock. Because mesh routers are much simpler. most existing multicomputers use mesh rather than torus networks. In addition. mesh networks allow channels twice as wide as those of torus networks under constant wire bisection limitation. Mesh networks also allow easy connection of I/O devices through edges which are not connected to any neighbor nodes. Through the edges. I/O devices can be easily connected. On the other hand, the primary drawback of mesh networks is that they utilize network channels unevenly.

Dally [14] introduced the express cube, an extension of low-dimensional k-ary n-cube networks. An express cube network consists of a hierarchy of mesh or torus networks superimposed on each other. The idea behind the express cube is to provide shortcuts for messages traveling long distances. Express cubes are embedded into basic mesh or torus networks, but the higher levels are much more sparsely populated. A message destined to a far node is routed through high-level channels instead of being routed through all of the intermediate nodes. The average latency may be reduced significantly by using the express cube network. However, express cubes seem to introduce a new problem, reduced bisection bandwidth. Many messages moving from a node in one half of a network to a node in the other half must get through the express cubes. The sparsely located express cubes may not provide enough bisection bandwidth. which may reduce sustainable peak throughput. Due to the limited bisection bandwidth, the performance of express cube networks is highly sensitive to the locality of applications. Express cube networks are motivated for pin-limited networks rather than wire-limited networks. For such networks, higher-dimensional networks are also interesting.

2.1.5 *WK*-Recursive Networks

The performance of an interconnection network is inversely proportional to its diameter. However, as mentioned earlier, there is a trade-off between the diameter and the degree of a network. Furthermore, due to technological limitations, the degree of a node which is the number of links branching off from the node has to be small (few units) and is fixed. This means that the scaleability of the network has to be independent of the node degree. In this section, we introduce a family of hierarchical interconnection networks, *WK*-Recursive, which have fixed node degree and are highly scalable [46].

Topology Description

A network of k nodes each of degree k can be fully connected still having k free links which can be viewed as being virtually similar to each component node of degree k. This structure can be used as a building block to construct larger structures recursively. In particular, a fully connected configuration composed of k of these virtual nodes (i.e. $k \times k$ real nodes) again offers k free links and reproduces, at a higher abstraction level, the virtual node structure. By recursively applying this technique, we can get a family of highly scalable, regular topologies, called *WK*-Recursive [46].

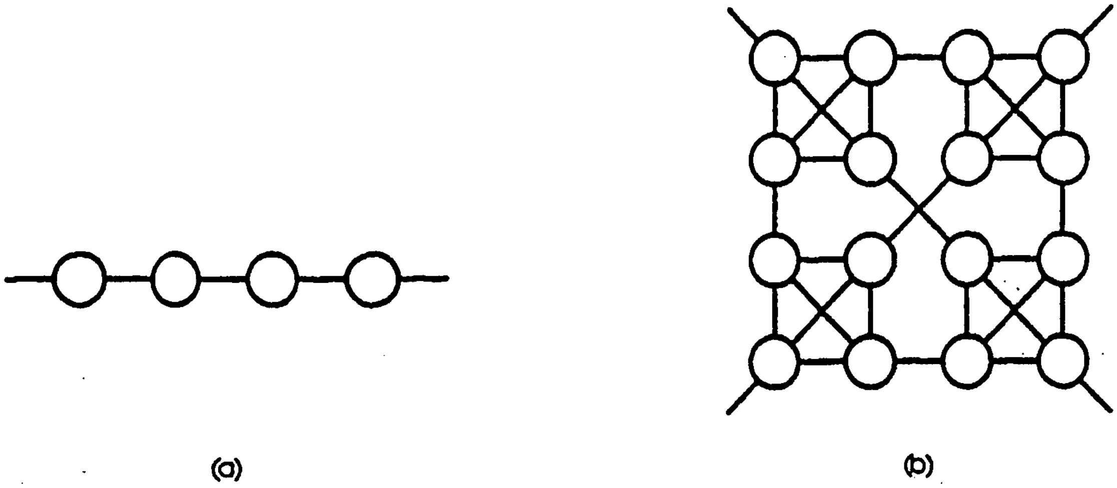

Figure 2.3: *WK*-Recursive topologies. (a) $k = 2$, $\ell = 2$, $N = 4$; (b) $k = 4$, $\ell = 2$, $N = 16$.

In a *WK*-Recursive network, if N signifies the number of real nodes, k the node degree, and ℓ the expansion level, we have

$$\ell = \log_k N \tag{2.3}$$

Figure 2.3 illustrates two examples of the topology with different values of N, k, and ℓ. The above expression permits to simply define indices for characterizing topologies belonging to the *WK*-Recursive class. For example, the diameter of the network is given by

$$D = 2^{\ell} - 1 \tag{2.4}$$

As we can see, the diameter depends only on the expansion level and is independent of the degree of the network. However, the bisection width of the network η is

$$\eta = \begin{cases} k/2 & \text{for } k = \text{even} \\ (k^2 - 1)/2 & \text{for } k = \text{odd} \end{cases} \tag{2.5}$$

and is independent of ℓ. This is a major disadvantage for WK-Recursive networks because it prevents the designer from trading off the width of a channel against the diameter of the network. However, if compare a WK-recursive network with $k = 4$ and a k-ary 2-cube, although both have 4 links per node, the diameter of the WK-recursive network is smaller than that of the k-ary 2-cube. If N = power of 4 for both networks, the diameter of the WK-recursive network is half as much as the diameter of the k-ary 2-cube, which is very attractive.

Deterministic Routing in WK-Recursive Networks

The routing scheme devised here, is a very simple algorithm which is valid for the whole class of WK-Recursive topologies since it does not depend on either the node degree or the expansion level of the structure. If we consider a first level W-wide virtual node, we can give each real node an index $n_0 \in \{0, 1, \ldots, W - 1\}$. Likewise, each of the first-level virtual nodes constituting a second-level virtual node is given an index n_1. In a network expanded at level ℓ composed of N real nodes each of them is characterized by an ℓ-tuple $(n_0, n_1, \ldots, n_{\ell-1})$. If we assign a weight to each index according to the expression,

$$n = \textstyle\sum_{i=0}^{\ell-1} n_i W^i \qquad \text{resulting} \qquad n \in \{0, \ldots, N - 1\}$$

each node will be uniquely identified by a node number n which can be regarded as the decimal coding of the W-ary number $n_W = n_\ell \ldots n_2 n_1$.

Each real node has k bi-directional links through which communications take place. Since $k = W$, $k - 1$ of the links are used to connect the node with the remaining $W - 1$ nodes in the first level and one is left free. The $k - 1$ links are numbered according to the value of the index n_0 of the node they are connected to, and the link which is left free is given the number equal to the value of the index n_0

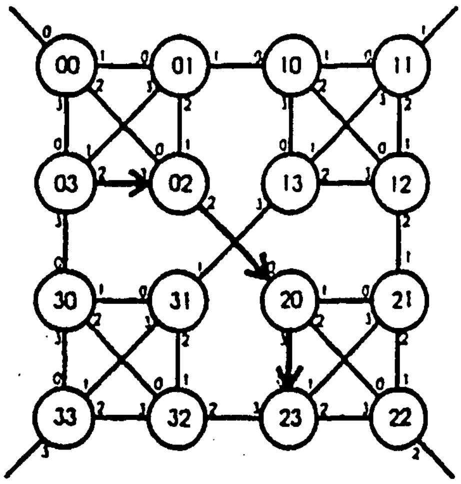

Figure 2.4: The routing from node 03 to node 23 in a network with $k = 4$, and $\ell = 2$.

of the node it belongs to. Figure 2.4 shows the routing from node 03 to node 23 in a network with $k = 4$, and $\ell = 2$.

When a message is sent from a node to another, the address of the destination node is included in the message in the W-ary notation $d_W = d_\ell \ldots d_2 d_1$. If the message arrives at a transit node t the routing takes place as follows:

```
if ($t_W = d_W$)
    the message has reached the destination
else
    forward the message through the link whose number is equal
    to the most significant digit of $d_W$ which is different from $t_W$
endif
```

2.2 Flow Control

Flow control is the resource management policy that is used to allocate communication resources (i.e. wires and buffers) to information units, *messages*, *packets*, and *flits*. Communication between nodes is performed by sending messages. A message may be broken into one or more packets for transmission. A packet is the smallest unit of information that contains routing information. A packet contains one or more flow control digits or flits. A flit is the smallest unit on which flow control is performed.

In multicomputer networks, if the source and destination of a message are not directly connected, the message is routed via other connected nodes. Among different switching models, *store-and-forward* or *packet-switching* is the most commonly used model. In this model, a packet is completely buffered before being passed to the next node. The communication latency of this model is a linear function of the number of hops the message has to traverse. In a network with channel bandwidth B, the latency of a message of length L traveling a distance of D hops using store-and-forward can be expressed as,

$$T_{sf} = (L/B)\,D \tag{2.6}$$

Newer multicomputers use *wormhole routing*, where wires and buffers are allocated to flits significantly smaller than an entire packet. The header flit or flits contains the routing information and the other flits just follow the header in a pipeline fashion. The communication latency of this model can be expressed as,

$$T_{wh} = \left(\frac{L_h}{B}\right) D + \frac{L - L_h}{B} \tag{2.7}$$

where L_h is the length of the header flit.

Performing flow control on units smaller than packets reduces latency, as shown in Figure 2.5. In store-and-forward routing, the total latency is the product of the length of the packets and the number of hops the packet has to travel. In wormhole routing, the total latency becomes instead the sum of the two quantities. The latency is reduced substantially for messages that traverse more than one channel and their

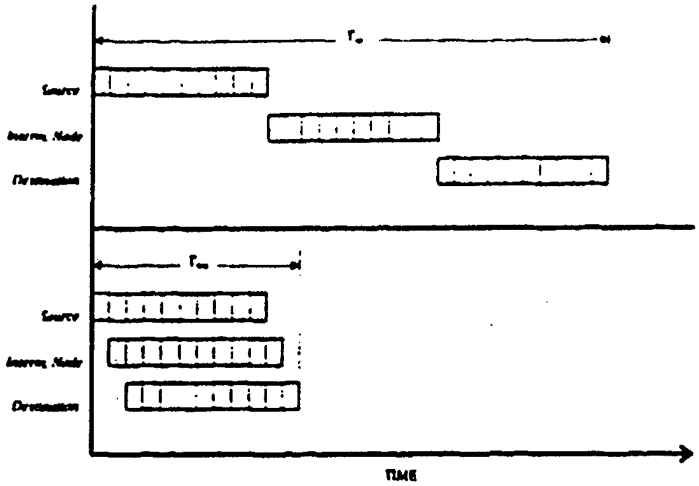

Figure 2.5: Latency of store-and-forward routing (top) versus wormhole routing (bottom).

lengths are long compared to the message distances. If the message length is very long, the latency becomes relatively insensitive to the distance which reduces the importance of message locality. It allows nonlocal communication to be used without incurring much degradation in message latency in an environment that operates under moderate traffic density [11. 49].

A hybrid strategy, *cut-through* [30], allocates storage buffers to packets as in store-and-forward. but pipelines the transmission of flits as in wormhole. In wormhole routing, the head of a packet will be immediately forwarded along its route whenever there is no conflict in channel access, or when the channel becomes idle. When a channel access conflict occurs, the packet is blocked behind the busy channel, waiting for it to become available. The body of the packet occupies the channels along its route, whereas the tail of the packet releases these occupied channels as it makes its way toward the destination. In cut-through routing, packets behave exactly as they do in the wormhole technique, as long as no channel access conflict occurs. However, when the requested channels are busy, the entire packet will be stored in the intermediate node at the collision spot.

In addition to *buffering* and *blocking* methods just explained, other schemes have also been proposed. One of these methods is *dropping* that is implemented on the BBN butterfly in which the second packet is allowed to continue advancing, but its flits are not stored as they arrive at the node. Another method is *misrouting* in which the other packet is routed to an idle but incorrect channel and from there is sent to the destination [38].

2.3 Routing

Routing is the method implemented to guide a message from a source to a destination in a network. A routing algorithm is a routing function $\mathbf{R}: N \times N \rightarrow C$ that maps the current node n_c and destination node n_d to the channel c_n on the route from n_c to n_d, $\mathbf{R}(n_c, n_d) = c_n$. Routing algorithms can be classified as *deterministic*, *oblivious*, or *adaptive*.

Most existing multicomputer networks [27, 29, 42] use deterministic routing. With deterministic routing, the path followed by a packet is determined solely by its source and destination. If any channel along this path is heavily loaded, the packet will be delayed. If any channel along this path is faulty the packet cannot be delivered. A common deterministic routing algorithm is dimension-order routing for k-ary n-cubes, where the packet is routed in one dimension at a time, arriving at the proper coordinate in each dimension before proceeding to the next dimension.

In an oblivious routing, the algorithm may choose different paths through the network, but may use no information about the network state in choosing the path. Randomized routing is an instance of oblivious routing in which each message is sent to a randomly chosen node, which then forwards it to its final destination. One

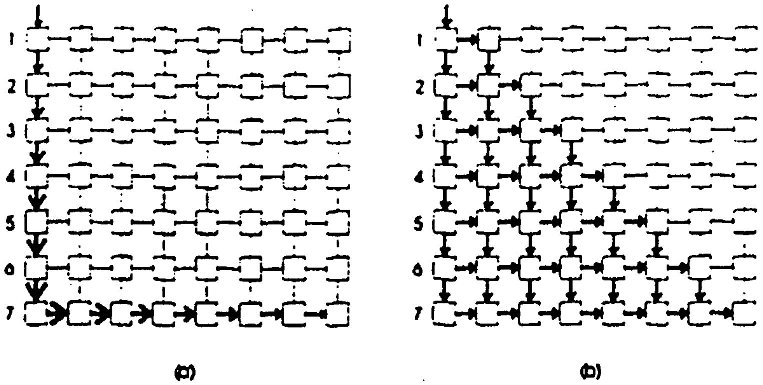

Figure 2.6: Load distribution under different routing schemes. (a) Deterministic: (b) Adaptive.

important disadvantage of randomized routing is that it does not preserve the locality of communications which limits the system scaleability.

Adaptive routing uses information about the state of the network to route the message. In this scheme, packets are detoured to other available paths as local congestion occurs in the network. Adaptive routing will eliminate hot-spots in the network traffic by distributing the load throughout the entire network. To illustrate how adaptive routing can improve the performance of an interconnection network. Figure 2.6 shows an 8×8 mesh in which the node at $(i,0)$ sends a packet to the node at $(7, i)$ for $i \in [0,7]$. With dimension-order deterministic routing [Figure 2.6 (a)], seven of the eight packets must traverse the channel from (6,0) to (7,0). Thus, only one of these seven packets can proceed at a time. With adaptive routing [Figure 2.6(b)] all of the packets can proceed simultaneously using alternate paths.

Furthermore, adaptive routing enhances the reliability of the system by taking advantage of the inherent path redundancy in the richly-connected multicomputers and circumventing faults in the network. Figure 2.7 demonstrate the advantage of adaptive routing to dimension-order-routing in handling failures.

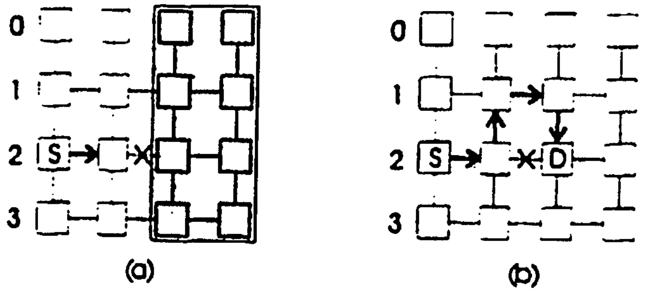

Figure 2.7: Failure handling under different routing schemes. (a) Dimension-order: (b) Adaptive.

2.3.1 Deadlock Avoidance

The flow control discipline must allocate resources to packets in a manner that avoids deadlock. Deadlock can occur when there is a cyclic dependency for resources. If two packets each hold resources required by the other to move, both packets will be blocked indefinitely. To avoid deadlock. the resources for which the packets are competing, have to be identified and a mechanism has to be introduced for breaking cyclic dependencies on the resources. There are different methods to approach this problem:

Structured Buffer Pools: These deadlock-free routing algorithms have been developed for store-and-forward computer communications networks [23, 26. 44, 45]. In these algorithms the message buffers in each node of the network are partitioned into classes, and the assignment of buffers to messages is restricted to define a partial order on buffer classes.

Turn Model: The algorithms employing this method [24] break the cyclic dependencies in the network graph by disallowing certain combinations of turns in the routing. For example, in one of these protocols called *Negative-first*, the turns from positive to negative directions (north to west and east to south) are made illegal. A big disadvantage of this protocol is its inefficiency in utilizing the entire network even when there is no deadlock.

Virtual Channels: In this method. each physical channel. $c_i \in C$. in the network is composed of one or more virtual channels. $c_{ij} \in C'$. the virtual channels associated with a single physical channel share physical channel bandwidth. allocated on a flit-by-flit basis. However. each virtual channel contains its own queue and is allocated on a packet-by-packet basis independently of the other virtual channels. Each virtual channel is considered logically a separate channel.

The use of virtual channels to construct deadlock-free routing functions is motivated by the definition of a routing function that maps $C \times N$ to C. rather than the conventional definition of a routing function that maps $N \times N$ to C [18]. By including C in the domain of the routing function. we explicitly define the dependencies between channels. These dependencies are represented by a *Channel Dependency Graph D.*

Definition 1 *A channel dependency graph D for a given interconnection network I and routing function* $\mathbf{R}$. *is a directed graph.* $D = G(C, E)$. *The vertices of D are the channels of I. The edges of D are the pairs of channels connected by* $\mathbf{R}$:

$$E = \{(c_i, c_j) \mid \mathbf{R}(c_i, n) = c_j \textit{ for some } n \in N\}$$

Since channels are not allowed to route to themselves, there are no 1-cycles in D. A necessary and sufficient condition for deadlock-free routing is that D be acyclic [18]. Figure 2.8 illustrates the application of virtual channels to a 4-loop. Since there is only one way to route around the network, a cycle exists in the channel dependency graph. The existence of this cycle makes it possible for the network to

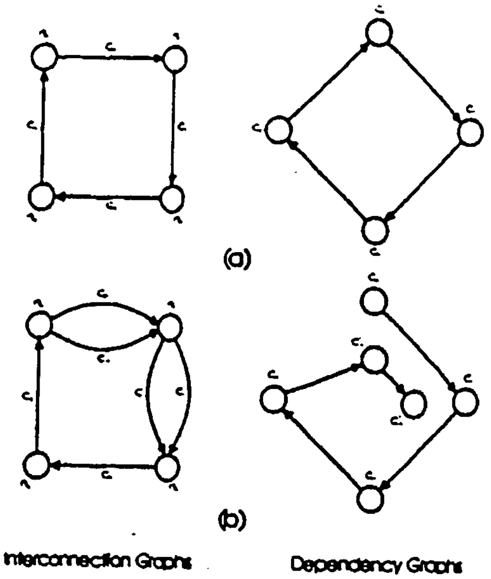

Figure 2.8: Breaking deadlock by adding virtual channel. (a) Original; (b) Deadlock free.

deadlock. To eliminate this cycle, we remove the 3-tuplets $(c_3, n_i, c_0) : i = 1, 2$ which removes the edge (c_3, c_0) from the dependency graph. To reconnect the network, we add two new virtual channels c_0' and c_1'. These channels are in parallel and share physical channels with c_0 and c_1.

2.4 The Software Messaging Layer

The performance of a parallel machine critically depends on the end-to-end cost of communication mechanism which is a combination of the routing time, the time to get messages into and out of the network, and software protocol overhead. Despite the advances in software messaging techniques, the software overhead still contributes the most to this total cost.

Messaging layers provide high-level communication abstractions required by the user applications which are not provided by the underlying network hardware. These communication services provided by the messaging layer releave the user from explicit network management. The most important services provided by the messaging layers are message delivery, message ordering, deadlock/overflow safety, and reliable delivery [28].

Future parallel machines will present certain characteristics such as arbitrary delivery order, finite buffering, and fault-detection (not fault-tolerance) which will have significant impact on the software messaging layers. Arbitrary delivery order of messages is usually caused by multipath routing (adaptivity) [20], virtual channels [16, 24], and time-sharing and process migration. In multipath routing different parts of the message take different routes and might reach the destination out of order. In timesharing the network state is swapped and later resumed in a state which may not preserve the delivery order. Finite buffering necessitates flow control to avoid deadlock by ensuring that there is always enough space at the nodes to store packets. Finally, lack of error-correction requires very reliable message delivery.

A messaging layer accommodates the services required by the user which are not supplied by the network. To provide message ordering in a network, which does not preserve delivery order, the messaging layer generally sequences and reorders the packets with a sequence number and buffering out-of-order packets. To provide deadlock and overflow safety in a machine with finite buffering the messaging layer preallocates storage for transmission of packets. Finally, the messaging layer introduces fault-tolerance into the network by source-buffering of message data and also requiring acknowledgments from destination to manage the finite buffers.

A study by Karamcheti and Chien [28] shows that even in a very efficient messaging layer, such as the Active Messages Layer on the CM-5, upto 50-70% of the software cost of the messaging can be attributed to providing end-to-end flow control, in-order delivery, and reliable transmission services.

2.5 Failure Handling

The fault-tolerance of a network depends directly on the number of alternate paths between its nodes, as long as there exists a routing algorithm which can take advantage of these multiple paths. A connected network with faulty links and/or nodes is called an *injured network*. To enable communication between non-faulty nodes, in an injured network, the information on component failures has to be made available to non-faulty nodes so as to route around the faulty components. This information can be kept at each node, or be added to the message. Clearly, if a node is equipped with the information about all the faulty components, it can easily route the message to the destination through the shortest distance. However, it is very costly, both in space and time, to provide the information on all the faulty components to every node on the network. Therefore, it is essential to develop routing schemes which can route messages in injured networks with the minimum information.

The fault-diagnosis in a multiprocessor system can be local or global. In global schemes, the information about the faulty components has to be distributed among the processors in the network. For example, Armstrong and Gray [5] propose an efficient algorithm for broadcasting the failure news to all the nodes in a hypercube.

We present two adaptive routing algorithms for GHCs and *WK*-Recursive networks which assume local fault-diagnosis. To make a node in the system capable of performing this diagnosis, we can use Asynchronous Communication Protocol between the neighboring nodes in which the sender waits for an acknowledgment from the receiver. A watch-dog timer generates a time-out interrupt, if the acknowledgment is not received after certain period. At this point, the sender can either try again, or just assume that the link to the neighbor is faulty and try to route the message through another link. Since a link in multicomputers connects only two nodes (no buses) assuming a link is faulty or assuming that the neighbor itself is faulty are equivalent.

Using asynchronous communication protocol between neighboring nodes in a multicomputer system has the advantage of making the fault-diagnosis transparent to the routing algorithm. The routing algorithm can simply assume that each node is aware of the status of its own links.

CHAPTER 3
EVALUATION FRAMEWORK

The performance of a network can be evaluated through analytical modeling or simulation. Contrary to common belief. these two methods do not replace each other and provide complementary means to understand and evaluate the network behavior.

Throughout this dissertation. whenever possible. we model networks analytically and use the developed models to investigate the effect of different parameters on the network performance. Derivation of an analytical model allows us to gain an in-depth understanding of the behavior of the network. and also observe the system response to specific conditions quickly and easily. We use simulation to inspect the validity of our analytical models. Although simulation does not present the behavioral patterns of the network as well as the analytical modeling does. it represents an accurate demonstration of the network behavior under a specific configuration.

However. certain network architectures. such as networks implementing wormhole switching or adaptive routing, are very difficult or even impossible to characterize analytically. The strong channel interactions and coupled event transitions of wormhole switching and the state dependent behavior of adaptive routing makes analysis of these networks impossible unless major simplifications and approximations are introduced. Unfortunately, in many cases, these simplifications eliminate the critical performance characteristics of the network. Consequently, we use simulation as the principal tool to evaluate the performance of these networks.

This chapter describes the framework in which we evaluate the performance of the interconnection networks. In section 3.1. we present the network node model which we have adopted throughout our study. The node architecture in the simulator is developed based on this model. In section 3.2. we describe the RSIM simulator developed at the High-Speed Digital Architecture Laboratory (HSDAL) which realizes the network model in detail. Section 3.3 discusses some of the assumptions we made for simple but fair comparison of the network simulations. This section also describes the various traffic loads we created to verify our analytical models with the simulation results. In section 3.4. we present some of the most important performance metrics u.ed to evaluate multicomputer networks. In the next chapters. we will provide and discuss results of the simulations.

3.1 Evaluation Model

The performance of a network depends heavily on the resource arbitration policies such as routing, switching, and queueing adopted by the network architecture. Support for multiple classes of traffic at a low architectural level requires careful analysis of the influence of these policies on the interaction between traffic classes. To evaluate the design options for the network, requires the ability to vary low level architectural parameters in a single unified framework.

Figure 3.1 presents the model of a node in the network. The model contains a computational unit and a communication unit. Although the simulator is capable of simulating both units, in this study, our focus is on the latter. When a message is initiated at a node, it is stored as a collection of packets in the message buffer. Each node has only a single message buffer which holds all the messages initiated from that node. The router inspects the messages in the buffer, based on a specified order, and

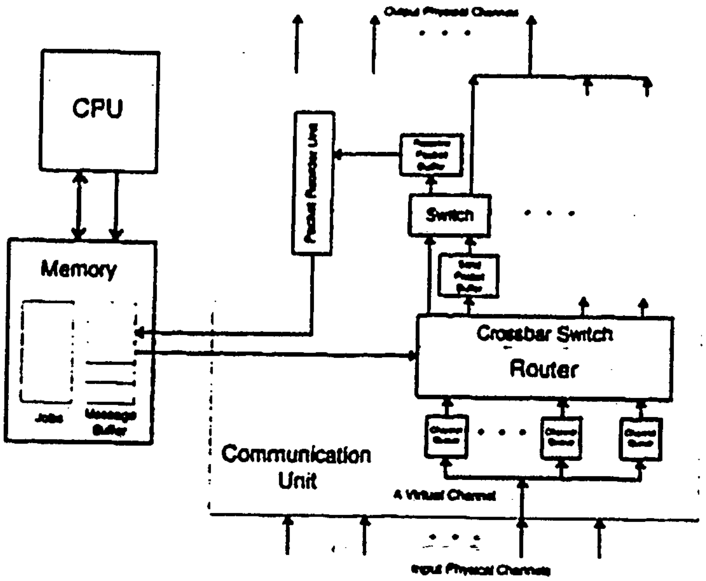

Figure 3.1: The model of a node in the simulator.

routes the packets through different channels. Each channel contains multiple virtual channels which are multiplexed at a granularity of one flit. Every virtual channel has a *send* and a *receive* packet buffer which hold the packets at the source and the destination, respectively. The packet flits are buffered in the channel queues while they pass through intermediate nodes. The number of flits stored in the channel queues depends on the adopted switching policy.

When a node receives the header flits of a packet, it decides whether to buffer the packet in the receive packet buffer, to forward the packet to the channel queue on the next node, or stall the packet. This decision is based on the destination address, the routing algorithm, the switching scheme, and the state of the buffers and the queues. By treating outbound virtual channels as individually reservable resources, the model can invoke a variety of routing and switching schemes through flexible control over reservation policies. The routing algorithm selects candidate outgoing virtual channels, while the switching scheme determines whether or not an incoming packet waits to acquire a selected outgoing virtual channel or buffers. Once a packet reserves an outgoing virtual channel. it competes with other virtual channels for access to the physical link. through an arbitration policy. The model includes several arbitration policies. including round-robin and priority-driven scheme.

3.2 The Simulator

The network model is evaluated using the RSIM simulator. RSIM is a simulation environment for studying different aspects of multicomputer networks. RSIM which is implemented in C++ allows evaluating the interconnection topology. interprocessor routing, communication flow control. application partitioning, and job allocation.

RSIM simulates different topologies such as. hypercube. mesh. torus. *WK*-Recursive. and user-defined structures. Each node has a communication unit and a computation unit operating simultaneaously. A Multiple-Program-Multiple-Data (MPMD) execution model is provided by allowing different jobs to run on different nodes. Nodes communicate by passing messages through the channels in the interconnection network. The simulator supports wormhole. virtual cut-through. and store-and-forward. as well as hybrid schemes. each under a variety of routing algorithms. Each physical channel can contain multiple virtual channels which are multiplexed based on user-specified schemes. This feature allows simulation of routing algorithms which use virtual channels to prevent deadlock. Different channel widths, queue sizes, and number of virtual channels for each individual link can be assigned in the simulation.

The router in the communication unit is a separate unit which can use different adaptive, deterministic, and random routing routines to direct the messages from a source to a destination. The communication can be simulated in flit- or packet-level

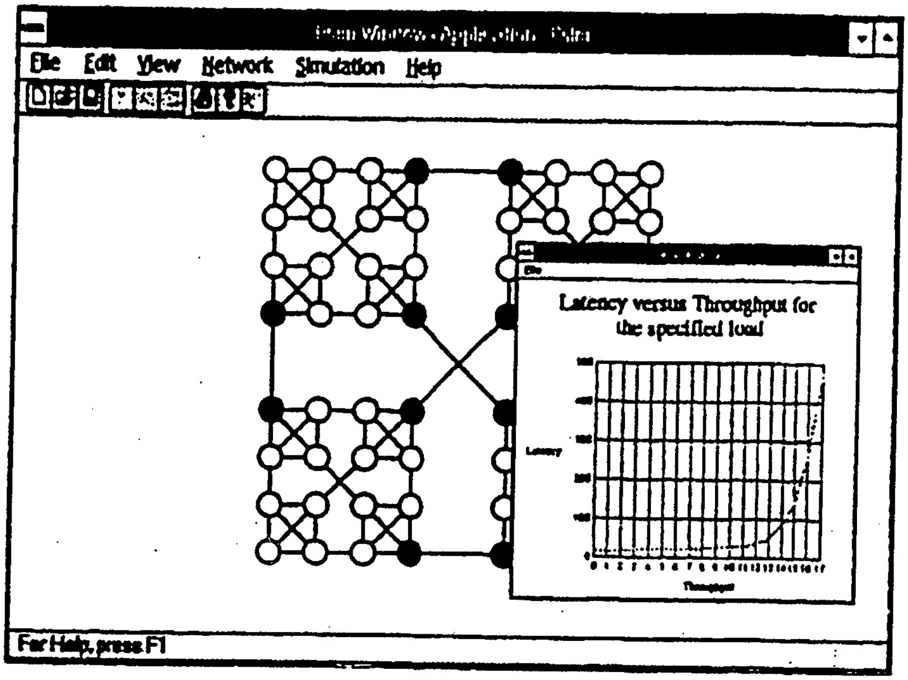
File Edit View Network Simulation Help
Latency versus Throughput for the specified load
Latency
Throughput
For Help, press F1

Figure 3.2: The user interface on the Windows version of the simulator.

applying user-defined flit or packet sizes. Permanent and transient failures of nodes and channel links can be easily simulated. Different loads such as *uniform-random destination*, *bit-reversal*, and *transpose* using *exponential* or *uniform* random inter-arrival times, are simulated by running predefined jobs on the nodes which generate such loads.

During the simulation, the necessary data is stored dynamically in a very efficient manner. The simulator can use this data to animate the simulation and also present histograms and graphical representations of the desired performance metrics, such as throughput and latency, after the simulation. Both graphical menu-driven and command-line user interfaces are implemented to allow the user to input different configurations for the system. The results of the simulation can be observed through graphs or files (Figure 3.2).

A parser is implemented to parse the configuration commands from a command file. Using a high-level specification language, the user can define the network topology and its configuration, the routing and flow-control policies, the traffic patterns generated by each node, and also the simulation control parameters. Appendix A describes the format of this command file.

To evaluate traffic mixing, the simulator associates each traffic class with a particular routing algorithm and switching scheme on a set of virtual channels. The tool includes an extensible set of routing-switching algorithms that interact with the router model through a well-defined set of instructions. This enables the specification of the routing-switching schemes to be separate from the router model. The algorithms can formally query the status of the router in order to execute state-dependent routing and switching decisions.

3.3 Performance Measurement

The workloads that we use to evaluate interconnection networks can vary in four dimensions: *traffic patterns*, *message size distribution*, *generation distribution*, and *deadline distribution*. Traffic patterns are the pairs of nodes that communicate and are described by message sources and destinations. We use uniform random destination selections for most of our simulation unless when we study the network under locality consideration. Message size distributions determine the size of each communication. The effect of varying message sizes can be as great as that of the traffic pattern. To show the effect of message size on the performance of the network, we try different constant message sizes in the simulation. Generation distribution, or arrival distribution, is the probability of a new message being injected into the network at each simulation cycle and is often normalized as a fraction of network bisection bandwidth. In most cases we use Poisson distribution for the message injection. We also study the network performance under bursty message injections. The interarrival time between bursts is exponentially distributed. The Deadline distribution is used to evaluate real-time systems and is the distribution of message deadlines. In most of our simulations we assume a constant *laxity* or a constant deadline distribution for the messages. We also examine the performance of the networks under exponentially distributed laxities.

To produce meaningful results. we allow simulations. using pseudo-random generation. to gather statistics over multiple runs. using different seeds for the random number generator. We try our best to gather statistics when the network is in steady state.

In most of the simulations performed for this research. unless stated otherwise. the simulator generates 2000 initial packets and then generates fluff packets until all the 2000 packets are collected. At this point. the system has usually reached a stable state and 5000 packets are generated which will be used to collect the performance data. Fluff packets are generated until all the 5000 packets are collected.

3.4 Performance Metrics

The penetration of parallel systems into on-line transaction processing and multi-media applications increases the importance of a variety of performance metrics. Network performance under load can be characterized by different performance metrics, *latency*, *throughput*, and *loss ratio*. Latency is measured from the time a message is generated to the time that the tail flit of the message is delivered at the destination node. If, due to resource conflicts, a generated message cannot be injected into the network immediately, the message is queued and the waiting time in the source queues is included in the latency. We report the average latency, latency standard deviation, and latency coefficient of variation. The former is traditionally used for best-effort traffic. The last two are often used to evaluate performance of the network under guaranteed or real-time traffic.

Throughput is another important metric of network performance and is defined as the total number of messages the network can handle per unit time. One method of estimating throughput is to calculate the capacity of the network, Γ, which is the total number of messages that can exist in the network at the same time. The maximum throughput of the network is typically some fraction of its capacity. The network capacity per node is the total bandwidth out of each node divided by the average number of channels traversed by each message.

While throughput at saturation is typically reported, other throughput measures are also relevant. First, throughput beyond saturation is an important characteristic for network stability. Ideally, throughput should be maintained even if the network is overloaded. Second, throughput fairness both over time, and spatially over different points in the interconnect can be essential to good performance. For example, one of the problems of the mesh network is that even with fair local arbitration at each router, it is much more difficult to get throughput going across the center of the mesh, than going away from it. Finally, to ensure timely completion of communication, supporting real-time or fault-tolerant computation, nodes may require a guarantee of throughput. Thus, the ability to provide such guarantees, absolute or statistical, is an important performance attribute of networks.

Two most important performance metrics of real-time systems are *loss ratio* and *guarantee ratio*. Loss ratio is the fraction of the number of lost packets over the number of total arrivals. Guarantee ratio is the ratio of the number of accepted packets to the number of acceptable packets. In real-time systems. the main objective of the scheduler is to ensure that the time requirements of the packets are met. or to guarantee this statistically by ensuring that the packets meet their deadlines with a high bounded probability. A real-time scheduling algorithm is often called optimal if its guarantee ratio is one. i.e.. it can find a feasible schedule whenever such schedule exists. Algorithms like *rate monotonic* [35], *earliest deadline first* (EDF) [19]. and *minimum laxity first* (ML) [19] are shown to be optimal in this sense. However. in designing real-time systems. we are particularly interested in minimizing the long-term loss ratio [50].

CHAPTER 4
DEVELOPMENT OF THE ANALYTICAL MODELS

The communication in multicomputer networks is directly affected by the availability of communication resources. channels and buffers. and how these resources are being managed. The variables which affect this resource management either belong to the architecture of the network or are the characteristics of the load which is applied to the network. The architectural variables are the channel switching scheme. number of virtual channels. channel arbitration scheme. routing, and network physical characteristics such as. dimension. channel width. node and wire delay. The load characteristics are the message inter-arrival time distribution. the message length and the communication locality. In this chapter. we develop analytical models which reflect the effect of these variables on the latency and throughput of the network.

4.1 Network Architectural Parameters

The performance of a multicomputer network depends heavily on the design constraints such as limits on the channel width. node size, and bisection width. As indicated in Chapter 2, the node size constraint is caused by the physical limit on the number of pins or connections to the chip or the board which contains the node. and the limit on bisection width is caused by the area constraint.

In this section, we will analyze networks comparatively based on the above constraints. We will find closed-form expressions for the network latency under each switching scheme considering the effects of switch and wire delays. as well as the contention. We will use these models to find the effective latency for the special cases of k-ary n-cube. generalized hypercube. and WK-Recursive networks. These models will be validated through comparisons with the simulation results. We will use the models to study the effect of other architectural and load parameters on the network performance.

4.1.1 The Model for Effective Latency

The latency of a packet through a network is not only affected by the adopted switching scheme. but it also depends directly on the network physical parameters and the load characteristics. The network physical parameters include the cycle time of a channel transfer. T_{chan}. and the channel-width. W. The load is characterized by the message length. L. the number of hops the message has to travel. n_{hops}. and the message arrival rate, λ. The latency through a network employing store-and-forward switching can be expressed as

$$\tau_{sf}(L,\lambda) = T_{chan}\left(n_{hops}\left(\frac{L}{W} + w_{sf}\right)\right) \tag{4.1}$$

where w_{sf} is the waiting time for a packet due to contention in a node and itself is a function of the load characteristics, the routing, and the topology. Similarly. the latency of a cut-through packet is

$$\tau_{ct}(L,\lambda) = T_{chan}\left(n_{hops}\left(\frac{L_h}{W} + w_{ct}\right) + \frac{L - L_h}{W}\right) \tag{4.2}$$

with L_h signifies the header length. When the network approaches saturation, the w term in both equations dominate the remaining terms and the latency basically becomes $n_{hops} \times w$. In the cut-through switching, the waiting time, w, depends on the buffer size, and as the buffer size grows, w approaches the waiting time exhibited by the store-and-forward switching. In virtual-cut-through which employs buffers larger than the size of a packet, $w_{ct} = w_{sf}$ and as we will observe in the next chapter, as the throughput is increased, due to the dominance of the w terms, the latency curves for the store-and forward and virtual-cut-through merge.

Setting w_{sf} and w_{ct} equal to zero yields the zero load latency for each switching which reflects the effect of the switch and wire delays on the overall latency. The length of a channel transfer can be minimally set equal to the sum of the switch delay and the delay through the longest wire in the network. In the past, wire delays have usually been ignored, due to their lower magnitude compared to the switch delays. However, the advances in technology are improving the delay in switches while wire delays have stayed almost constant. Soon, switches and wires with similar dimensions will have comparable delays [1].

Although we assume the clock cycle is equal to the sum of the switch delay and the longest wire delay, it is important to note that, the influence of long wires on the clock cycle can be mitigated by introducing multiple clock transmissions on longer wires or, as it is done in wide area networks, allowing multiple bits to be in flight on the wire at any given time. In this case, the channel propagation delay, T_{prop}, will be a function of the wire length, while the channel transmission cycle, T_{chan}, will be less than T_{prop} and stays constant. In equation 4.2, T_{prop} will be the coefficient of the first term in the bracket and T_{chan} will be the coefficient of the second term. In our analysis we assume $T_{chan} = T_{prop}$; however, the effect of $T_{chan} \neq T_{prop}$ can be easily studied by changing the message length by a factor $\frac{T_{chan}}{T_{prop}}$.

Since networks are embedded in a two or maximum three dimensional space. networks of higher dimension create uneven wire lengths. In a network. the ratio of the longest wire to the shortest wire. α_w. is usually a function of the network topology. In our analysis. we assume the delay of the shortest wire to be unity. We also let the switch delay be greater than this delay by a constant factor. α_s.

If D_{avg} signifies the average distance a packet has to travel. the equations for the average latencies become

$$\overline{T}_{sf}(L,\lambda) = (\alpha_s + \alpha_w)\left(D_{avg}\left(\frac{L}{W} + \overline{w}_{sf}\right)\right) \tag{4.3}$$

$$\overline{T}_{ct}(L,\lambda) = (\alpha_s + \alpha_w)\left(D_{avg}\left(\frac{L_h}{W} + \overline{w}_{ct}\right) + \frac{L - L_h}{W}\right) \tag{4.4}$$

As noted before. in both store-and-forward and virtual cut-through. which possess packet-sized or larger channel queues. the contention parameter of the delay. w. is basically the same. On the other hand. in networks implementing cut-through switching with smaller than packet-sized queues. such as wormhole. a blocked packet occupies multiple channels and contributes to the contention on all those channels simultaneously which creates a completely different type of waiting time distribution. Therefore, depending on whether the size of the channel queue is larger than the size of a packet or not, two different approaches have to be adopted to find the distribution of the waiting time.

Initially, we will find a model for the traffic in direct networks with larger than packet-sized queues. We start by deriving an expression for the delay in a switching node considering flit-sized packets. We are assuming the size of a flit is equal to the width of a channel; consequently, a flit is transferred in one cycle over the channel. Although, the assumption on flit-sized packets makes the communication indifferent

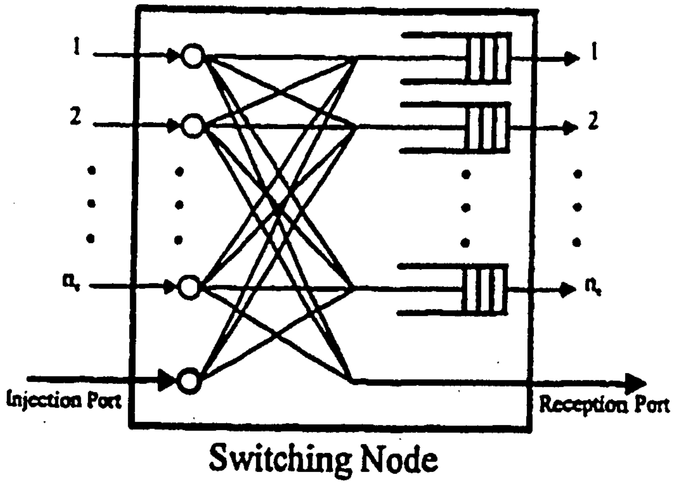

Figure 4.1: Model for a switching node.

to the switching scheme. later. we will extend the analysis to include larger packets which makes the waiting time depend on how the packets are switched in the network. For our analysis. we also assume that each node has n_c network inputs and n_c network outputs. In direct networks. each node also has an input port and an output port connected to the processor in that node. We refer to the former as the injection port and to the latter as the reception port (Figure 4.1).

Kruskal and Snir in [32] derived a model for the contention in buffered multistage interconnection networks. An important difference between these networks and direct networks, which we are considering, is that the packet arrivals at a switching node in a multistage interconnection network have a simple binomial distribution. However, in direct networks the distribution of packet arrivals depends on the topology and the routing. In our analysis we follow some of the steps that they took to derive the latency model. We assume that a queue of unbounded capacity is associated with each output port and in each cycle a flit-sized packet leaves the queue. We assume that the network is synchronous, so that packets can be sent only at the end of each cycle. We signify the number of packets arrived at each output queue in a cycle n with ν_n. Since at each node there are n_c network inputs and one injection port, for a total of $n_c + 1$ inputs, in each cycle, upto $n_c + 1$ packets can join each output queue. If we let the random variable q_n represent the number of packets in a queue in a cycle n, we will have

$$q_{n+1} = \begin{cases} q_n - 1 + \nu_{n+1} & \text{for } q_n > 0 \\ \nu_{n+1} & \text{for } q_n = 0 \end{cases} \qquad (4.5)$$

It is important to notice that even when the queue is completely empty, an incoming packet has to get queued for , at least, one cycle. In other words, the service time is a part of the queueing time; and to find the waiting time, we have to subtract the service time (in our case 1) from the queueing time.

Equation 4.5 resembles the recurrence equation which describes the number of customers in an $M/G/1$ queue [31]. To find the average number of customers in an $M/G/1$ queue, the well-known *Pollaczek-Khintchine (P-K) mean value formula* is used. We follow the path taken by Kleinrock [31] to derive the P-K formula and diverge when he includes the assumption on a Markovian arrival distribution. Additionally, we derive the second moment for the distribution of the number of packets in an output queue which will be used to derive the variance of the waiting time for a packet in an output queue. We are interested in the variance of the waiting time distribution to evaluate the predictability of the system.

We use the shifted discrete step function defined as

$$U_k \overset{\text{def}}{=} \begin{cases} 1 & \text{for } k > 0 \\ 0 & \text{for } k \leq 0 \end{cases} \qquad (4.6)$$

and by applying it to Equation 4.5 we can express the number of packets in an output buffer with a single recurrence equation.

$$q_{n+1} = q_n + \nu_{n+1} - U_{q_n} \tag{4.7}$$

Taking the expectation of both sides of the equation. we will have

$$E[q_{n+1}] = E[q_n + \nu_{n+1} - U_{q_n}] = E[q_n] + E[\nu_{n+1}] - E[U_{q_n}] \tag{4.8}$$

If $\tilde{q}$ signifies the limiting distribution of the random variable q_n and the system is ergodic [36] (a reasonable assumption.) the jth moment of q_n exists in the limit as $n \rightarrow \infty$ and is expressed as

$$\lim_{n\rightarrow\infty} E[q_n^j] = E[\tilde{q}^j] \tag{4.9}$$

Applying this to equation 4.8 yields the limiting distribution. namely.

$$E[\tilde{q}] = E[\tilde{q}] + E[\tilde{\nu}] - E[U_{\tilde{q}}] \tag{4.10}$$

or

$$E[\tilde{\nu}] = E[U_{\tilde{q}}] \tag{4.11}$$

Now, if we square both sides of equation 4.7, we have

$$q_{n+1}^2 = q_n^2 + \nu_{n+1}^2 + U_{q_n}^2 + 2q_n\nu_{n+1} - 2q_nU_{q_n} - 2\nu_{n+1}U_{q_n} \tag{4.12}$$

$U_{q_n}^2 = U_{q_n}$ and $q_nU_{q_n} = q_n$ and also $q_n\nu_{n+1}$ and $\nu_{n+1}U_{q_n}$ are products of two independent variables. Applying these to equation 4.12 and forming expectations of both sides, we get

$$E[q_{n+1}^2] = E[q_n^2] + E[\nu_{n+1}^2] + E[U_{q_n}] + 2E[q_n]E[\nu_{n+1}] - 2E[q_n] - 2E[\nu_{n+1}]E[U_{q_n}] \quad (4.13)$$

Using equation 4.9, in the limit as $n \rightarrow \infty$, we have

$$E[\tilde{q}^2] = E[\tilde{q}^2] + E[\tilde{\nu}^2] + E[U_{\tilde{q}}] + 2E[\tilde{q}]E[\tilde{\nu}] - 2E[\tilde{q}] - 2E[U_{\tilde{q}}]E[\tilde{\nu}] \quad (4.14)$$

Applying equation 4.11 and the equality $Var[\tilde{\nu}] = E[\tilde{\nu}^2] - E[\tilde{\nu}]^2$ to the above equation, we get

$$2E[\tilde{q}](1 - E[\tilde{\nu}]) = Var[\tilde{\nu}] + E[\tilde{\nu}] - E[\tilde{\nu}]^2 \quad (4.15)$$

Therefore, the average number of packets in an output buffer will be

$$E[\tilde{q}] = \frac{Var[\tilde{\nu}]}{2(1 - E[\tilde{\nu}])} + \frac{E[\tilde{\nu}]}{2} \quad (4.16)$$

To find the variance of the packet waiting time in an output buffer we have to find the second moment of $\tilde{q}$. In order to find $E[\tilde{q}^2]$, we cube both sides of equation 4.7.

$$\begin{aligned} q_{n+1}^3 &= q_n^3 + \nu_{n+1}^3 - U_{q_n}^3 - 6q_n\nu_{n+1}U_{q_n} + 3q_n^2\nu_{n+1} + 3q_n\nu_{n+1}^2 \\ &\quad - 3q_n^2U_{q_n} + 3q_nU_{q_n}^2 - 3\nu_{n+1}^2U_{q_n} + 3\nu_{n+1}U_{q_n}^2 \end{aligned} \quad (4.17)$$

We form the expectations of both sides of equation 4.17 considering $U_{q_n}^3 = U_{q_n}^2 = U_{q_n}$ and $q_nU_{q_n} = q_n$ and also $q_n\nu_{n+1}$ and $\nu_{n+1}U_{q_n}$ are products of two independent variables. We will have

$$E[q_{n+1}{}^3] = E[q_n{}^3] + E[\nu_{n+1}{}^3] - E[U_{q_n}] - 6E[q_n]E[\nu_{n+1}] +$$

$$3E[q_n{}^2]E[\nu_{n+1}] + 3E[q_n]E[\nu_{n+1}{}^2] - 3E[q_n{}^2] +$$
$$3E[q_n] - 3E[\nu_{n+1}{}^2]E[U_{q_n}] + 3E[\nu_{n+1}]E[U_{q_n}] \qquad (4.18)$$

To get the limiting distribution, we allow $n \to \infty$. Applying 4.9 and 4.11, we get

$$\begin{aligned} E[\tilde{q}^3] &= E[\tilde{q}^3] + E[\tilde{\nu}^3] - E[\tilde{\nu}] - 6E[\tilde{q}]E[\tilde{\nu}] + 3E[\tilde{q}^2]E[\tilde{\nu}] + \\ &\quad 3E[\tilde{q}]E[\tilde{\nu}^2] - 3E[\tilde{q}^2] + 3E[\tilde{q}] - 3E[\tilde{\nu}^2]E[\tilde{\nu}] + 3E[\tilde{\nu}]^2 \end{aligned} \qquad (4.19)$$

Simplifying the above equation yields

$$E[\tilde{q}^2] = \frac{E[\tilde{\nu}^3] - E[\tilde{\nu}] - 3E[\tilde{\nu}^2]E[\tilde{\nu}] + 3E[\tilde{\nu}]^2 + E[\tilde{q}](3 + 3E[\tilde{\nu}^2] - 6E[\tilde{\nu}])}{3(1 - E[\tilde{\nu}])} \qquad (4.20)$$

If we know the first and the second moments of the packet arrival distribution, $\tilde{\nu}$, we can use equations 4.16 and 4.20 to obtain the values for $E[\tilde{q}]$ and $E[\tilde{q}^2]$.

Now that we have derived the first and second moments for the number of packets in an output queue, we can use this information to get the first and the second moments of a packet queueing time distribution. By *Little's Law*, the average number of customers in a queueing system is equal to the average arrival rate of customers times the average time each customer spends in the queue. In our case, the average arrival rate of packets to an output buffer is $E[\tilde{\nu}]$, and also, based on our assumption, the service time for each packet, x is one cycle. As mentioned before, the service time is included in the queueing time, s; therefore, the average waiting time for a packet in an output buffer will be

$$\overline{w} = E[w] = E[s] - 1 = \frac{E[\tilde{q}]}{E[\tilde{\nu}]} - 1 = \frac{Var[\tilde{\nu}]}{2E[\tilde{\nu}](1 - E[\tilde{\nu}])} - \frac{1}{2} \qquad (4.21)$$

To determine the second moment of the queueing time distribution, we look more closely at how the expectations for queueing time distribution and the queue length distribution are evaluated. The waiting time is in terms of the channel cycle time and the values that it takes can only be integer multiples of the channel cycle time. Since, the service time of a packet, $\bar{x}$, is one cycle, the number of packets in a queue can directly correspond to the waiting time of the packets. However, the number of packets in a queue is evaluated throughout the entire time while the waiting time is only defined when the queue is not empty. In an empty queue, there is no packet and waiting time has no meaning. We know that in a single-queue output, the channel utilization, $\rho = E[\tilde{\nu}].\bar{x}$, is the probability that the channel is busy. Since $\bar{x} = 1$, $E[\tilde{\nu}]$ signifies the percentage of the time that there is a packet in the queue keeping the channel busy. We can use this to find a relation between the expectations of $\tilde{q}$ and the time spent in the queue,s.

$$E[s^j] = \frac{E[\tilde{q}^j]}{E[\tilde{\nu}]} \tag{4.22}$$

As we can see, equation 4.22 for $j = 1$ yields the same result obtained by the Little's Law in equation 4.21. Since the service time is constant, its variance will be zero. Consequently, the variance of the queueing time for a packet will be equal to the variance of the packet waiting time. So, we have

$$Var[w] = Var[s] = \frac{E[\tilde{q}^2]}{E[\tilde{\nu}]} - \left(\frac{E[\tilde{q}]}{E[\tilde{\nu}]}\right)^2 \tag{4.23}$$

Therefore, if the first and the second moments of the packet arrival distribution, $\tilde{\nu}$, are known, we can derive $E[\tilde{q}]$ from 4.16 and $E[\tilde{q}^2]$ from 4.20 and use equation 4.23 to obtain the variance of the waiting time of a packet in an output channel queue.

To determine the average of the packet delay and its variance, we have to know the distribution of the random variable $\tilde{\nu}$, the number of arrived packets in an output queue during a cycle. Unlike indirect networks [32], in our networks $\tilde{\nu}$ does not have a simple binomial distribution and depends directly on the topology and the adopted routing algorithm. In the following sections we determine this distribution for k-ary n-cubes, generalized hypercubes, and WK-Recursive networks using both deterministic or adaptive routing.

4.1.2 Analysis of Latency in k-ary n-cubes

In a k-ary n-cube, with randomly chosen message destinations, the average number of hops a message has to travel, $D_{avg} = nk_d$, where k_d is the average distance a message must travel in each dimension. In a torus with unidirectional channels $k_d = (k-1)/2$. If the torus implements bidirectional channels, $k_d = k/4$ for even k, and $(k - 1/k)/4$ for odd k. In a mesh, which lacks the end-around wraps of a torus, $k_d = (k - 1/k)/3$.

Latency under Zero Contention

An important issue regarding k-ary n-cube networks is how to choose k and n for a network with a given number of nodes, N, to achieve the best performance. Without considering any implementation constraints, high-dimensional networks appear to perform better because of their lower diameter. A smaller diameter implies reduced latency and, more importantly, much less channel contention. Figures 4.2 and 4.3 show the latency graphs under no physical constraints for three different k-ary n-cubes using cut-through and store-and-forward switching, respectively. These graphs do not address the effect of the wire length on the latency.

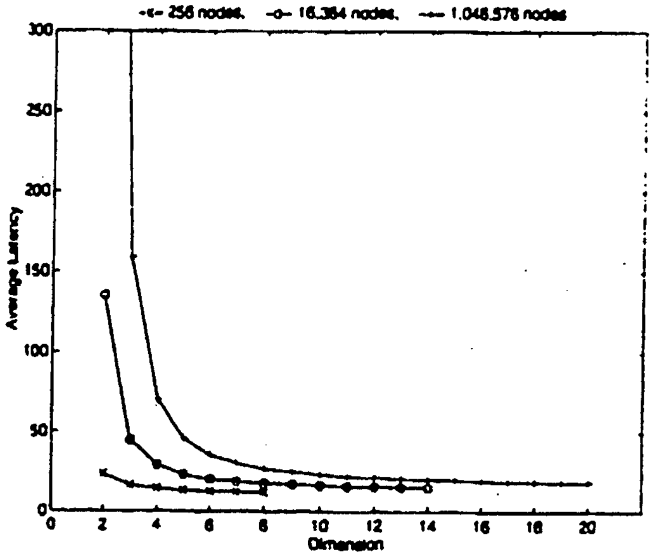

Figure 4.2: Average latency vs dimension using cut-through switching with no physical constraints. $L = 250$ bits.

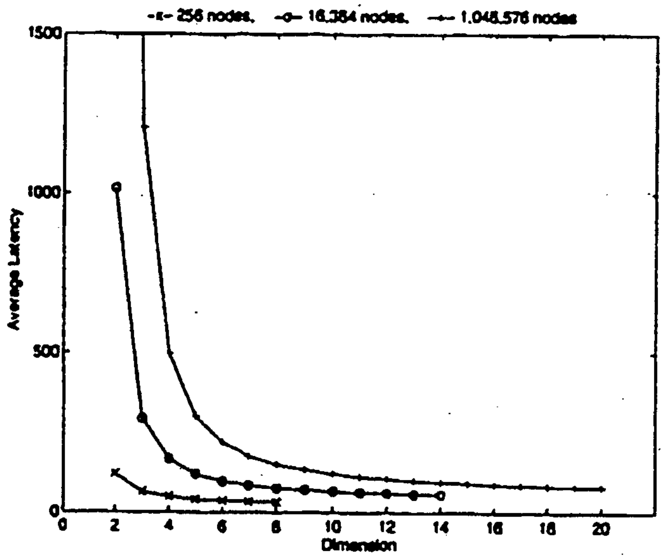

Figure 4.3: Average latency vs dimension using store-and-forward switching with no physical constraints. $L = 250$ bits.

However, the optimal choice of k and n critically depends on a variety of design constraints. One of these constraints is the limit imposed by the wire density in the network. As noted earlier, this limit can be represented by the network wire bisection width, η_w. For k-ary n-cube networks with wraparounds (torus), the bisection width is

$$\eta(k,n) = 2k^{n-1} = \frac{2N}{k} = \frac{2N}{\sqrt[n]{N}} \qquad (4.24)$$

If the wraparound connections do not exist (mesh), η is halved. For a fixed-size network (fixed N), as network dimension n grows, η increases superlinearly. Since the width of each channel is derived as $W = \frac{\eta_w}{\eta}$, for a fixed η_w, the channel width, W, decreases rapidly as the dimension n grows. For a given message length, narrow channels increase message latency, overwhelming the advantages of high-dimensional networks.

For a binary n-cube, $k = 2$ meaning $\eta_w(2,n) = WN$. In order to compare different k-ary n-cube networks under constant wire bisection width, we set η_w equal to N to normalize to a binary n-cube with unit-width channels, $W(k,2) = 1$. Therefore, the channel width $W(k,n)$ of a k-ary n-cube with the same bisection width will be

$$W(k,n) = \frac{\eta_w(2,n)k}{2N} = \frac{k}{2} \qquad (4.25)$$

Under this assumption, each processing node connects to $2n$ channels, each $k/2$ bits wide. Thus, the number of pins per node is $d_w = nk$. Figure 4.4 is the plot of the pin density as a function of dimension for k-ary n-cubes with three different numbers of nodes. Under the assumption of constant η_w, low-dimensional networks have the disadvantage of possessing more pins per node. However, with the increase in n, the

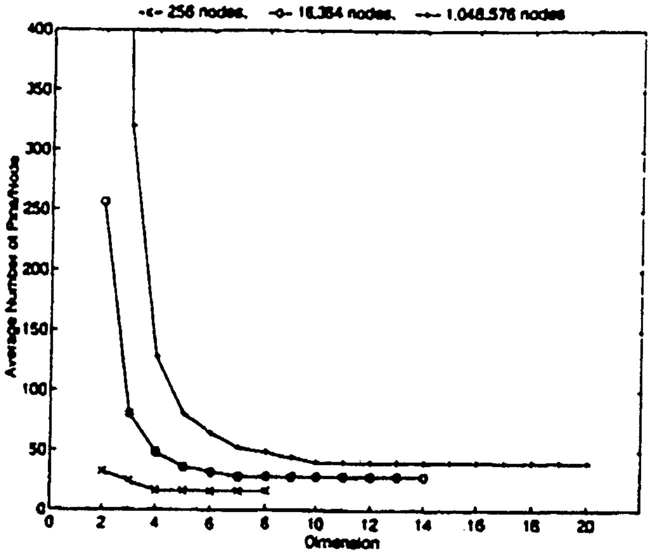

Figure 4.4: Pin density vs dimension assuming constant η_w.

number of nodes decreases very rapidly. This plot can be used to inspect if a network which yields low latency also possesses a reasonable number of pins per node.

Substituting D_{avg} for a torus and equation 4.25 into equation 4.4 and assuming a header length equal to the width of the channel. the zero-contention latency on a torus under cut-through switching will be

$$\overline{T}_{ct}(L,0) = (\alpha_s + \alpha_w)\left(\frac{n(k-1)}{2} + \frac{2L}{k} - 1\right) \qquad (4.26)$$

If the switching delay of the network dominates the wire delay. we can assume T_{chan} as not being a function of the topology and being constant. Substituting $k = N^{\frac{1}{n}}$ into 4.26 and assuming $T_{chan} = 1$, we get the latency equation under the switch delay dominance,

$$\overline{T}_{ct}(L,0) \approx \frac{n(N^{\frac{1}{n}} - 1)}{2} + 2LN^{\frac{-1}{n}} \qquad (4.27)$$

Figure 4.5 shows the average latency as a function of the dimension, assuming no contention, for the previous k-ary n-cubes. For these plots, we assume $L = 256$ bits

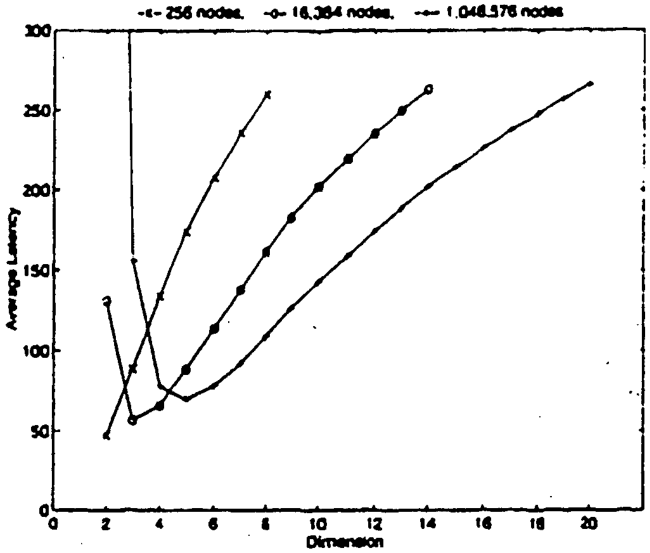

Figure 4.5: Average latency vs dimension using cut-through switching with constant η and constant delay.

and constant wire delay which represents the case when the switch delay dominates the wire delay.

As we can see from the plots, low dimensional networks achieve lower latency than higher dimensional networks. The minimum is generally achieved when the two terms of the latency are almost equal, or $D_{avg} \approx \frac{L}{W}$. For networks with few dimensions, the latency due to the distance dominates. As n is increased, latency is reduced until distance and aspect ratio, $\frac{L}{W}$, are equal. Beyond that point, the $\frac{L}{W}$ component of the latency dominates. This is in contrast with the latency curves under no physical constraints (Figure 4.2).

Following the same path, but using equation 4.3 instead of equation 4.4, we can write the zero-load latency equation under the store-and-forward switching. The average latency for store-and-forward under constant wire delay will be

$$\overline{T}_{sf}(L,0) = nL(1 - N^{\frac{-1}{n}}) \tag{4.28}$$

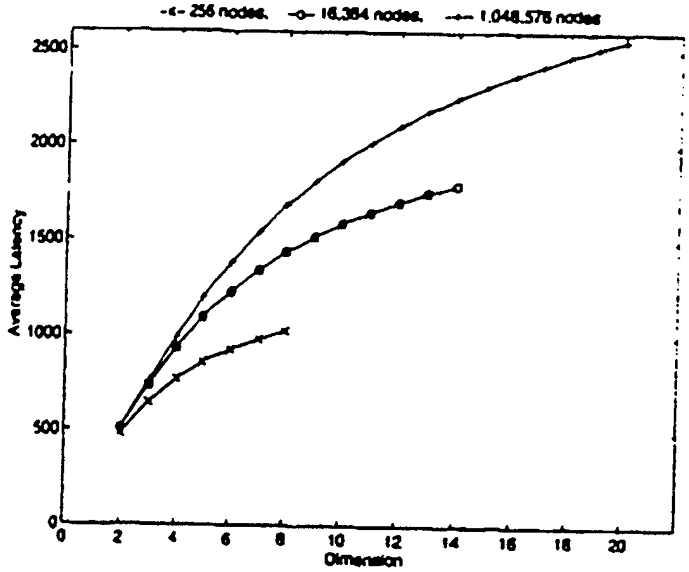

Figure 4.6: Average latency vs dimension using store-and-forward switching with constant η and constant delay.

Figure 4.6 shows the average latencies using the store-and-forward switching. In this case, the latencies monotonically increase as the network dimension increases and the networks with the fewest dimensions, due to their largest channel width, have the lowest latencies. This is in contrast with the latency curves under no physical constraints (Figure 4.3).

If a k-ary n-cube is embedded in a plane, $n/2$ dimensions of the network are laid out in each of the two physical dimensions. Since the total number of nodes $N = k^n$, each additional dimension contributes to $\sqrt{k}$ factor increase in the number of nodes in each physical dimension. This increase results in a $\sqrt{k}$ increase in the length of the longest wire. In other words, assuming planar mapping and linear wire delay, the ratio of the longest wire to the shortest wire in a k-ary n-cube will be the same as α_w and is given by

$$\alpha_w = (\sqrt{k})^{n-2} = N^{(\frac{1}{2}-\frac{1}{n})} \quad (4.29)$$

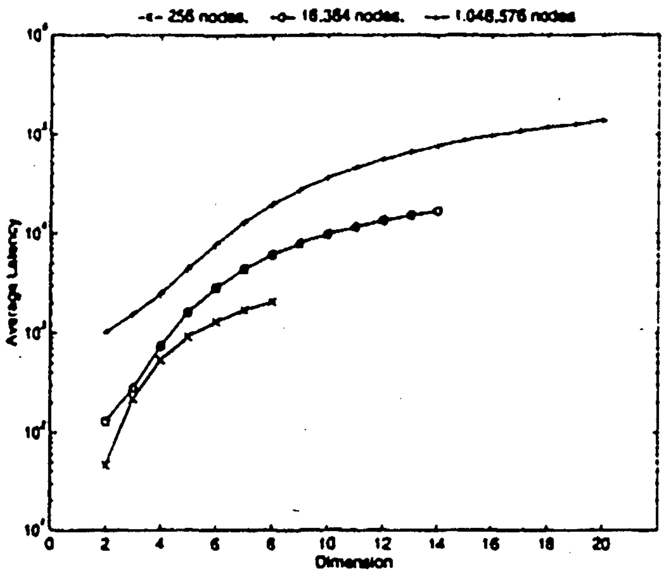

Figure 4.7: Average latency vs dimension using cut-through switching with constant η and linear wire delay.

Substituting 4.29 into equation 4.26 and simplifying the result will give us the equation for zero-load latency under cut-through switching for a k-ary n-cube with linear wire delay,

$$\overline{T}_{ct}(L.0) = \left(\alpha_s + N^{(\frac{1}{2}-\frac{1}{n})}\right)\left(\frac{n(N^{\frac{1}{n}}-1)}{2} + 2LN^{\frac{-1}{n}}\right) \qquad (4.30)$$

Figure 4.7 shows the average network latency as a function of dimension for the previous k-ary n-cubes, assuming linear wire delay and a message length. $L = 256$. and $\alpha_s = 0$. In this case, a two-dimensional network always gives the lowest latency. Under the linear delay assumption, latency is determined solely by the bandwidth and by the physical distance traversed; and a two-dimensional network offers the highest channel bandwidth and the most direct physical route.

As it was noted in chapter 2, for very short wires, the wire delay is a logarithmic function of its length, or $T_{prop} \propto 1 + \log \alpha_w$. Substituting this into 4.26 and some simplifications yields the equation for zero-load latency under cut-through switching

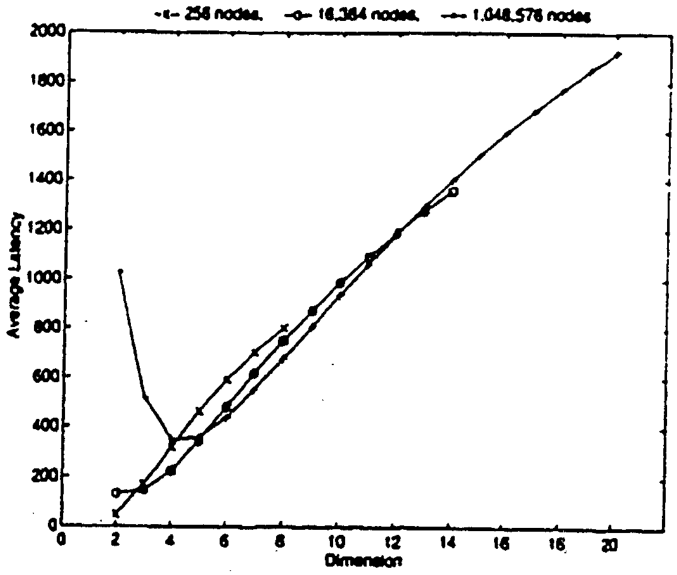

Figure 4.8: Average latency vs dimension using cut-through switching with constant η and logarithmic wire delay.

for a k-ary n-cube with logarithmic wire delay.

$$\overline{T}_{ct}(L,0) = \left(\alpha_s + \left(1 + \left(\frac{1}{2} - \frac{1}{n}\right)\log N\right)\right)\left(\frac{n(N^{\frac{1}{n}} - 1)}{2} + 2LN^{\frac{-1}{n}}\right) \tag{4.31}$$

Figure 4.8 shows the latency results for cut-through switching under logarithmic wire delay.

An important point we have noticed so far is that physical limitations favor low-dimensional networks. However, this is true even when there is no bisection width or node size constraint. This is due to the rapid increase in the wire length as the number of dimensions is increased.

<u>Assertion 1</u> *In a k-ary n-cube with a fixed number of processors, N, if the number of dimensions is increased from n to n + c, the wire length increases by a factor of* $N^{\frac{c}{n(n+c)}}$.

Proof: We know that $\frac{T_n^w}{T_2^w} = N^{\frac{1}{2}-\frac{1}{n}}$ where T_n^w is the length of the longest wire in an n-dimensional k-ary n-cube. Therefore.

$$\frac{T_{n+c}^w}{T_n^w} = \frac{\frac{T_{n+c}^w}{T_2^w}}{\frac{T_n^w}{T_2^w}} = \frac{N^{\frac{1}{2}-\frac{1}{n+c}}}{N^{\frac{1}{2}-\frac{1}{n}}} = N^{\frac{1}{n}-\frac{1}{n+c}} = N^{\frac{c}{n(n+c)}}$$

Analysis of Latency under Contention

If the k-ary n-cube implements dimension-order routing, the message moves towards the destination from one dimension to another in a fixed order. This causes the packets from different input channels at a node to be routed to an output channel with different probabilities. This effect becomes more important for networks with large index and consequently large k_d. In these networks. a message has to travel a longer distance in a dimension before switching to another dimension. This means that a message continues. on the average. more on the same channel than switching to other channels.

It is important to note again that since we assumed unit-sized packets with single cycle channel transmission, the probability of a packet arriving at an incoming channel (the arrival rate) will be the same as the channel utilization which signifies the probability of finding the channel busy. In other words

$$\rho = E[\tilde{\nu}].\bar{x} = E[\tilde{\nu}] \tag{4.32}$$

If the probability of a network request from a processor in a given cycle (the average injection rate) be λ_{inj} and each message on the average has to travel D_{avg} hops. for

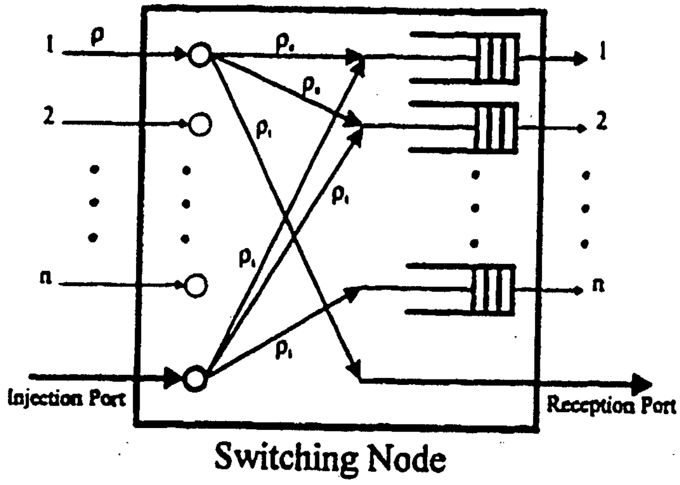

Figure 4.9: Switching probabilities on an input channel.

a node with n_c uni-directional channels. the channel utilization will be

$$\rho = \frac{\lambda_{inj} D_{avg}}{n_c} \tag{4.33}$$

The network bandwidth per node, or the message rate for which the network reaches saturation is obtained when $\rho = 1$. At this point, the throughput of the network reaches the network capacity, Γ, defined in the previous chapter. For a k-ary n-cube. $D_{avg} = nk_d$, and $n_c = n$: therefore the channel utilization will be

$$\rho = \frac{\lambda_{inj} n k_d}{n} = \lambda_{inj} k_d \tag{4.34}$$

We assume the routing probability of an incoming packet at a channel in a given dimension is nonnegligible only for the continuing channel in that dimension, and for the channel corresponding to one lower dimension. The larger the index of a network is, the more accurate this assumption will be. However, even for low index values it will generate acceptable results. We also assume that the injected packets at a node are steered to output ports randomly. Obviously, under dimension-order routing, this assumption is inaccurate and the probability that a packet is injected into a channel depends on the dimension of the channel. However. this assumption prevents the model to become too complex and. on the average. yields reasonably good results. Figure 4.9 shows the model of a node switch. The packet probability ρ in a channel along a given dimension is composed of three components. ρ_i. ρ_s. and ρ_c: ρ_i corresponds to the packets injected into a dimension from the processor at the node. or the packets received by the processor from that dimension. ρ_s signifies the packets that switch to a dimension. And finally. ρ_c corresponds to the packets which continue along the same dimension. As defined before. the probability a packet is injected by the processor at the node is λ_{in_j}. and the probability this packet is routed to a given output channel in the node is $1/n$. Therefore. $\rho_i = \lambda_{in_j}/n = \rho/nk_d$. ρ_i also signifies the probability a packet exits the network at a node.

Since a packet switches dimensions on the average once every k_d hops. the probability it will switch to one lower dimension in any given cycle is $\rho_s = (\rho - \rho_i)/k_d = \rho(1 - 1/nk_d)(1/k_d)$. A packet which stays in the network and does not switch has to continue in the same dimension, and therefore. $\rho_c = (\rho-\rho_i)(1-1/k_d) = \rho(1 - 1/nk_d)(1 - 1/k_d)$.

Assuming the three above probabilities for a packet at a switch, we can write the distribution for $\tilde{\nu}$ as

$$p(\tilde{\nu}) = \begin{cases} (1-\rho_i)(1-\rho_s)(1-\rho_c) & \tilde{\nu} = 0 \\ (1-\rho_i)(1-\rho_s)\rho_c + (1-\rho_i)\rho_s(1-\rho_c) + \rho_i(1-\rho_s)(1-\rho_c) & \tilde{\nu} = 1 \\ (1-\rho_i)\rho_s\rho_c + \rho_i\rho_s(1-\rho_c) + \rho_i(1-\rho_s)\rho_c & \tilde{\nu} = 2 \\ \rho_i\rho_s\rho_c & \tilde{\nu} = 3 \\ 0 & \tilde{\nu} > 3 \end{cases} \quad (4.35)$$

We find the expectation of this distribution as

$$\begin{aligned} E[\tilde{\nu}] &= \sum_{i=1}^{3} \tilde{\nu} p(\tilde{\nu}) \\ &= \rho_i + \rho_s + \rho_c \\ &= \rho \qquad (4.36) \end{aligned}$$

which matches with our previous result of equation 4.32. Also, the second moment of the distribution will be

$$\begin{aligned} E[\tilde{\nu}^2] &= \sum_{\tilde{\nu}=1}^{3} \tilde{\nu}^2 p(\tilde{\nu}) \\ &= \rho + 2\rho_c\rho_s + 2\rho_s\rho_i + 2\rho_i\rho_c \qquad (4.37) \\ &= \rho + \frac{2\rho^2}{k_d}(1 + 1/n) - \frac{2\rho^2}{k_d^2}(1 + 2/n + 1/n^2) + \frac{2\rho^2}{k_d^3}(2/n + 1/n^2) - \frac{2\rho^2}{k_d^4 n^2} \end{aligned}$$

and since $Var[\tilde{\nu}] = E[\tilde{\nu}^2] - E[\tilde{\nu}]^2$, the variance of $\tilde{\nu}$ can be approximated as

$$Var[\tilde{\nu}] \approx \rho - \rho^2 + 2\rho^2[\frac{1}{k_d}(1 + 1/n) - \frac{1}{k_d^2}(1 + 2/n)] \qquad (4.38)$$

Substituting equations 4.36 and 4.38 in 4.21, we get

$$\overline{w} = \frac{\rho}{(1-\rho)}[\frac{1}{k_d}(1 + 1/n) - \frac{1}{k_d^2}(1 + 2/n)] \qquad (4.39)$$

Similarly, the third moment of the distribution will be

$$\begin{aligned} E[\tilde{\nu}^3] &= \sum_{\tilde{\nu}=1}^{3} \tilde{\nu}^3 p(\tilde{\nu}) \\ &= \rho + 6\rho_c\rho_s + 6\rho_s\rho_i + 6\rho_i\rho_c + 6\rho_i\rho_s\rho_c \end{aligned}$$

$$= \rho + \frac{6\rho^2}{k_d}(1 + 1/n) - \frac{6\rho^2}{k_d^2}(1 + 2/n + 1/n^2) + \frac{6\rho^2}{k_d^3}(2/n + 1/n^2) - \frac{6\rho^2}{k_d^4 n^2} + \frac{6\rho^3}{n k_d^2}(1 - 1/k_d - 2/nk_d + 3/n^2k_d^2 - 1/n^2k_d^3) \quad (4.40)$$

Now that we have derived the first three moments of the arrival distribution of a k-ary n-cube, we can use the relations in the previous section and calculate the mean and the variance of the packet waiting time. However, upto this point, we assumed unit-sized packets and single-cycle packet transfers. We now extend the model to include packets of arbitrary size L. Each new packet takes $\frac{L}{W}$ cycles to transfer through a single link, or $\bar{x} = \frac{L}{W}$, where W is the width of the channel. To reflect this increase in the service, we increase the delay through the switch by a factor $\frac{L}{W}$, namely

$$\overline{w} = \frac{\rho L}{W(1-\rho)}[\frac{1}{k_d}(1 + 1/n) - \frac{1}{k_d^2}(1 + 2/n)] \quad (4.41)$$

On the other hand, the increase in service time will also increase the channel utilization represented by equation 4.32 by a factor $\frac{L}{W}$. The new channel utilization will be

$$\rho = \frac{L\lambda_{inj}k_d}{W} \quad (4.42)$$

To get the new relation for the variance of the waiting time, we use the relation $Var(Bw) = B^2Var(w)$, where $B = \frac{L}{W}$.

Validation of the Model

To verify our delay model we tested it through simulations against several network types and a variety of workload parameters. Our simulator generates packets of length L at every node with poisson distribution with a specified average rate. We configure the simulator to use dimension-order routing to steer the generated packets to randomly chosen destinations. The simulator collects the data to generate the required statistics such as the average latency, throughput, and channel utilization.

To be able to compare the model with the simulation results, we have to use the constant wire and node delay model. Using a constant cycle time of $T_{chan} = 1$, and $D_{avg} = nk_d$ the relation for the average latency under both switching times will be

$$\overline{T}_{ct} = nk_d\left(\frac{L_h}{W} + \frac{\rho L}{W(1-\rho)}[\frac{1}{k_d}(1 + 1/n) - \frac{1}{k_d^2}(1 + 2/n)]\right) + \frac{L-L_h}{W} \tag{4.43}$$

$$\overline{T}_{sf} = nk_d\left(\frac{L}{W} + \frac{\rho L}{W(1-\rho)}[\frac{1}{k_d}(1 + 1/n) - \frac{1}{k_d^2}(1 + 2/n)]\right) \tag{4.44}$$

We assume $L_h = W = 1$ which makes the packet length be L channel-widths. Figures 4.10 and 4.11 compare the network latency predicted by the model and through simulation for an 8×8 torus using packets with length $L = 16$ under both switching schemes. We can see that the latency predicted by the models are very close to the actual latency measured through the simulation. Both models underestimate the latency slightly at high loads which can be attributed to the adopted dimension-order routing which causes packets through higher dimension channels suffers higher than average delays.

To verify the assumption we made on the impact of the packet size on the latency, we ran simulation with different packet sizes on networks with different dimensions and radices. To focus on the effect of the packet size, we kept the injection rate at a constant level which yielded a channel utilization of 0.5. Figures 4.12 and 4.13 show the comparison of the results obtained from the simulation and the

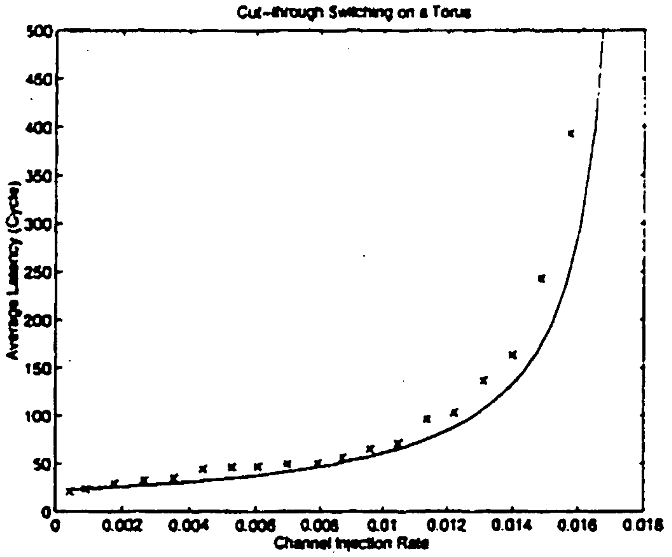

Figure 4.10: Comparing the model with the simulation under virtual cut-through switching. Dashed line correspond to the simulation results.

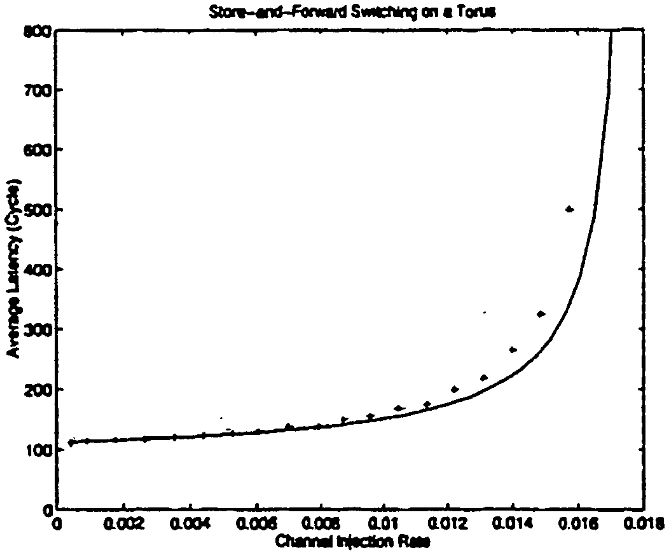

Figure 4.11: Comparing the model with the simulation under store-and-forward switching. Dashed line correspond to the simulation results.

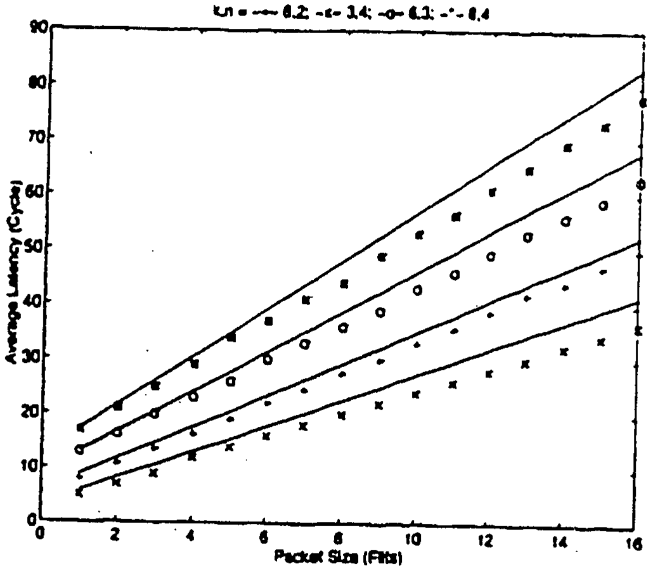

Figure 4.12: Studying the impact of the packet size on the latency. Comparison of the model with the simulation under virtual cut-through switching. Dashed line correspond to the simulation results.

models for each switching scheme. The depicted results demonstrate that the assumption we made, on the linearity of the latency with the packet size, was in fact an acceptable assumption.

4.1.3 Analysis of Latency in Generalized Hypercubes

In an n-dimensional generalized hypercube signified with the set of radices $\{k_n, \ldots, k_1\}$, the number of nodes, $N = \prod_{i=1}^{n} k_i$. Under randomly chosen message destinations, the average number of hops a message has to travel, $D_{avg} = \sum_{i=1}^{n} k_d^i$, where k_d^i is the average distance a message must travel in the i-th dimension. Since in a single dimension, using bi-directional channels, the distance between any two distinct nodes is one, the overall average distance in a dimension i will be: $k_d^i = \frac{k_i - 1}{k_i}$. Therefore, the average distance in a GHC with bi-directional channels is

$$D_{avg} = \sum_{i=1}^{n} \frac{k_i - 1}{k_i} \qquad (4.45)$$

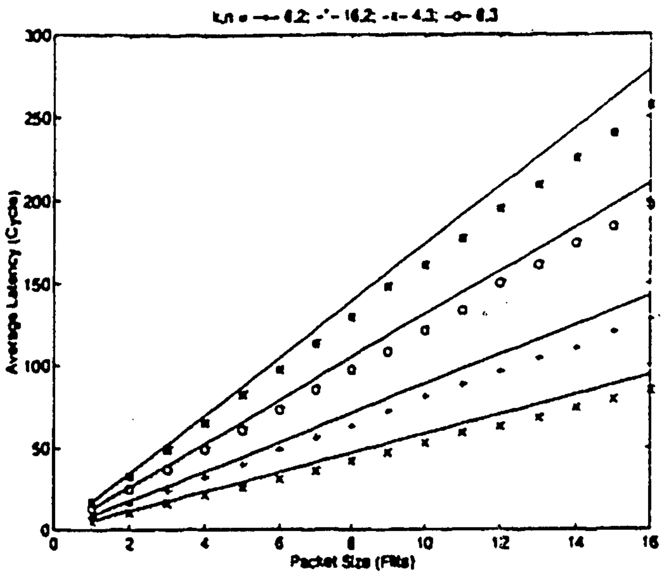

Figure 4.13: Studying the impact of the packet size on the latency. Comparison of the model with the simulation under store-and-forward switching. Dashed line correspond to the simulation results.

Latency under Zero Contention

The wire bisection width of a GHC with W-wide channels is

$$\eta_w = \eta \times W = \frac{NW}{k_{min}}\lceil\frac{k_{min}}{2}\rceil\lfloor\frac{k_{min}}{2}\rfloor \tag{4.46}$$

where k_{min} is the radix with the smallest magnitude. For simplicity, we assume that all the radices are even and equal. Under this assumption, $D_{avg} = \frac{n(k-1)}{k}$ and $\eta_w = \frac{NWk}{4}$. We will analyze the zero-load latency in a GHC with only linear wire delay under different constraints and switching schemes.

Similar to the k-ary n-cube, if a GHC is embedded in a plane, $n/2$ dimensions of the network are laid out in each of the two physical dimensions and each additional dimension contributes to $\sqrt{k}$ factor increase in the number of nodes in each physical dimension. This increase results in a $\sqrt{k}$ increase in the length of the longest wire. In other words, assuming planar mapping and linear wire delay, the ratio of the longest wire to the shortest wire in a GHC will also be the same as α_w and is given by

$$\alpha_w = (\sqrt{k})^{n-2} = N^{(\frac{1}{2}-\frac{1}{n})} \tag{4.47}$$

Considering cut-through switching and using the results above with the assumption of unit header length the latency equation is

$$\overline{T}_{ct}(L,0) = \left(\alpha_s + N^{(\frac{1}{2}-\frac{1}{n})}\right)\left(n(1 - N^{\frac{-1}{n}}) + \frac{L}{W}\right) \tag{4.48}$$

Let us initially analyze the network performance. assuming no contention and no physical constraint. such as wire density or pin numbers. With that assumption we allow a constant channel width over networks of all dimensions. We are aware that assuming a constant channel width. W. over all dimensions indirectly means that the size of the system can grow without any physical constraint bounding this growth which is something completely impractical. However, this study will allow us to study the effect of the signal propagation alone on the overall average latency.

Figure 4.14 shows the latency under the constant channel width constraint when $\frac{L}{W} = 24$.

Since for a binary n-cube with bi-directional channels $\eta_w = \frac{WN}{2}$. to compare different GHC networks under constant wire bisection width, we set η_w equal to $\frac{N}{2}$ to normalize to a bianry n-cube with unit-width channels, $W = 1$.

$$W(k,n) = \frac{4\eta_w(2,n)}{Nk} = \frac{2}{k} \tag{4.49}$$

Under this assumption, each processing node connects to $n(k-1)$ channels. each $2/k$ bits wide. Thus, the number of pins per node is $d_w = \frac{n(k-1)}{k}$. Figure 4.15 is the

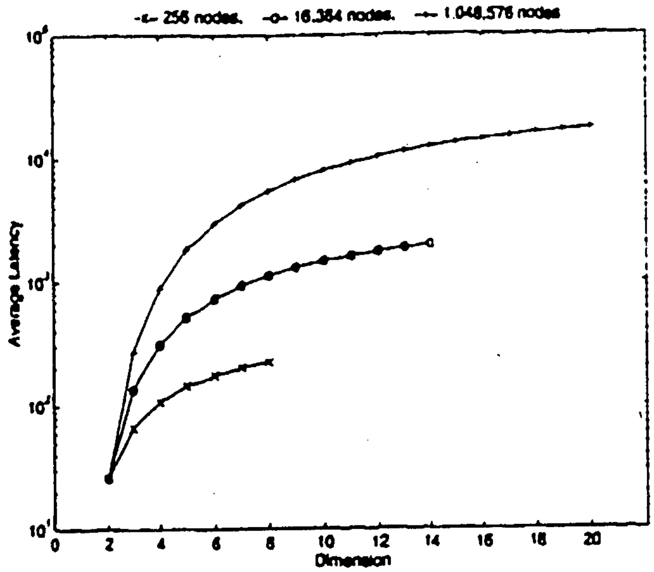

Figure 4.14: GHC average latency vs dimension using cut-through switching with constant $\frac{L}{W}$.

plot of the pin density as a function of dimension for GHCs with the three different numbers of nodes. As we can see from the figure. under the assumption of constant η_w, the number of pins per node monotonically increases as n increases. This is in contrast with the results obtained for the k-ary n-cubes. This plot can be used to inspect if a network which yields low latency. under the constant wire bisection width assumption, also possess a reasonable number of pins per node.

Analysis of Latency under Contention

In a GHC with dimension-order routing, the message moves towards the destination from one dimension to another in a fixed order. This causes the packets from different input channels at a node to be routed to an output channel with different probabilities. Despite this similarity with the k-ary n-cubes, in GHCs a message travels an entire dimension in maximum one hop. On the contrary, in k-ary n-cubes, a message has to travel a longer distance in a dimension before switching to another

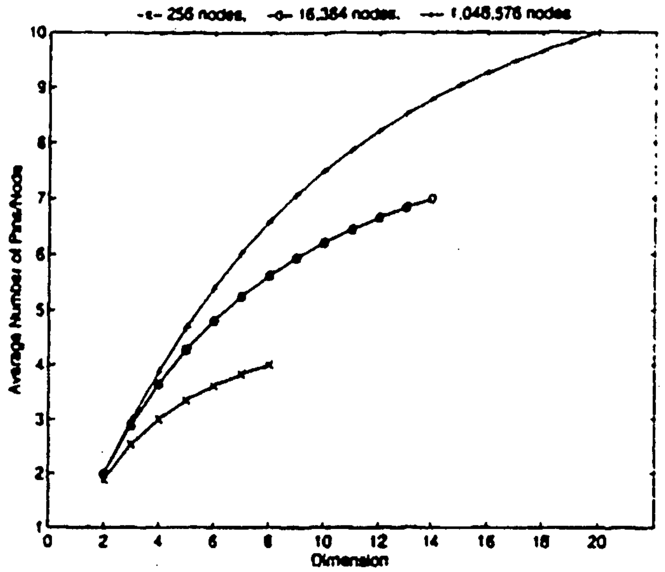

Figure 4.15: GHC pin density vs dimension assuming constant η_w.

dimension causing a message to continue, on the average, more on the same channel than switching to other channels.

In our analysis for the k-ary n-cubes, we ignored all except three routing probabilities of an incoming packet at a channel in a given dimension. This assumption which was mainly made to simplify the derived contention model, was more accurate for networks with large indices. However, in a GHC, a similar assumption will be invalid, because no matter how large the network index is, the entire dimension will be traversed in one hop.

In this section, we will find a model for the packet latency in a GHC considering the routing probabilities through all the channels. Again, initially we assume unit-sized packets and later extend the model to include larger packets. Due to the unit-sized packets and single cycle channel transmission, the probability of finding a channel busy (the channel utilization) is the same as the probability of a packet arriving at an incoming channel (the arrival rate).

We assume the traffic from each node is generated by a poisson process with arrival rate λ_{in_j}. Also. assuming the message destinations are uniformly distributed and independent. the average distance that a message has to travel will be $\frac{n(k-1)}{k}$ hops. Since each node has $n(k-1)$ bi-directional channels. the overall average channel utilization will be

$$\rho = \frac{\lambda_{in_j}\frac{n(k-1)}{k}}{n(k-1)} = \frac{\lambda_{in_j}}{k} \tag{4.50}$$

Later. we will prove that in fact the average utilization of each channel. independent of its dimension. is $\frac{\lambda_{in_j}}{k}$.

If you recall. in our analysis of k-ary n-cubes. we assumed that the injected packets at a node are steered to output ports randomly. Obviously. under dimension-order routing, this assumption is not completely accurate and the probability that a packet is injected into a channel depends on the dimension of the channel. Here. we abandon that assumption and find the exact routing probability of an injected message into each dimension.

When a packet is injected by a processor into the network. the packet is routed in the first dimension. unless both the source and the destination lie in the same coordinate at that dimension ($\frac{1}{k}$ probability). In other words. the probability that the message is routed into the first dimension is $\frac{k-1}{k}$. Since the node has $k-1$ channels connected in each dimension, the probability of a message being routed into one of the channels is $\frac{1}{k}$. In general, the probability that an injected message by a node is routed to a specific channel in the d-th dimension is $\frac{1}{k^d}$.

Since GHC is a symmetric topology, under uniform destination selection, we can model one of the nodes as a representative of all the nodes in the network. On this representative node, if ρ_d^i signifies the probability of packet arrivals at the i-th output channel in dimension d, signified by C_d^i where $1 \leq i \leq k-1$, we will prove that ρ_d^i is equal to the overall average channel utilization, ρ, obtained by equation 4.50.

<u>*Assertion 2*</u> *In an n-dimensional GHC with index k, if the probability that a node injects a packet into the network is λ_{inj}, the average probability of packet arrival at any output channel is $\frac{\lambda_{inj}}{k}$.*

Proof: We consider one of the $k-1$ output channels in dimension d, say channel C_d^i. Assuming the routing is done from low to high dimensions, the packets arrived at this output channel can come from any of the $(d-1)(k-1)$ input channels connected to the node in the first $d-1$ dimensions or from the injection channel in that node. In other words, ν, the number of packets arrived at an output queue, can acquire values ranging from 0 through $(d-1)(k-1)+1$. If the packet injection rate from the processor is λ_{inj}, the probability that an injected packet is routed to channel C_d^i is $\frac{\lambda_{inj}}{k^d}$. Similarly, the probability that a packet from any of the $k-1$ channels, say channel $C_j^{i'}$, in dimension j, where $j < d$, is routed to channel C_d^i is $\frac{\rho_j^{i'}}{k^{d-j}}$, where $\rho_j^{i'}$ is the packet arrival probability at the i'-th input channel in dimension j. Since $\rho_j^{i'}$ for all $1 \leq i' \leq k-1$ is the same, we show this probability byρ_j which represents the packet probability in only one of the channels in the j-th dimension. Therefore, the overall packet arrival probability for our output channel is

$$\rho_d = \frac{\lambda_{inj}}{k^d} + (k-1)\sum_{j=1}^{d-1} \frac{\rho_j'}{k^{d-j}} \tag{4.51}$$

Again, ρ_d represents the packet arrival probability at only one of the $k-1$ output channels in dimension d. We prove that if $\rho_j = \frac{\lambda_{inj}}{k}$ for $1 \leq j < \ell$, then $\rho_\ell = \frac{\lambda_{inj}}{k}$.

Using this and the fact that $\rho_1 = \frac{\lambda_{inj}}{k}$, we can prove that $\rho_d = \frac{\lambda}{k} = \rho$ for all d.

$$\begin{aligned}
\rho_d &= \frac{\lambda_{inj}}{k^d} + (k-1)\sum_{j=1}^{d-1}\frac{\lambda_{inj}}{k^{d-j+1}} \\
&= \lambda_{inj}\left[\frac{1}{k^d} + (k-1)\sum_{m=2}^{d}\frac{1}{k^m}\right] \\
&= \lambda_{inj}\left[\frac{1}{k^d} + \sum_{m=1}^{d-1}\frac{1}{k^m} - \sum_{m=2}^{d}\frac{1}{k^m}\right] \\
&= \lambda_{inj}\left[\frac{1}{k^d} + \frac{1}{k} - \frac{1}{k^d}\right] \\
&= \frac{\lambda_{inj}}{k} \qquad (4.52)
\end{aligned}$$

Using equations 4.50 and 4.52, we have proved that $\rho_d = \rho$ for any output channel. This is an important result meaning that the average packet arrival probability of all the channels are the same and equal to $\frac{\lambda_{inj}}{k}$.

4.1.4 *WK*-Recursive with Deterministic Routing

In symmetric networks such as tori, under randomly chosen message destinations, all the channels are utilized uniformly. In the previous section, this uniformity allowed us to model one node as a representative for all the nodes in the network. In *WK*-Recursive networks, if message destinations are chosen randomly, different links will have different throughput. Figure 4.16 depicts the number of packets transferred through each link in a (3,3)-WKR when each node sends messages to all the other nodes. As the figure shows, with identical channels, the unbalance of traffic creates bottlenecks in the network. In a (W, L)-WKR these bottlenecks happen at the links which connect the $(W, L-1)$-WKR clusters. These links convey the largest amount of traffic and are highlighted in Figure 4.16. The ratio of the messages passed through these links to total messages passed through all channels is

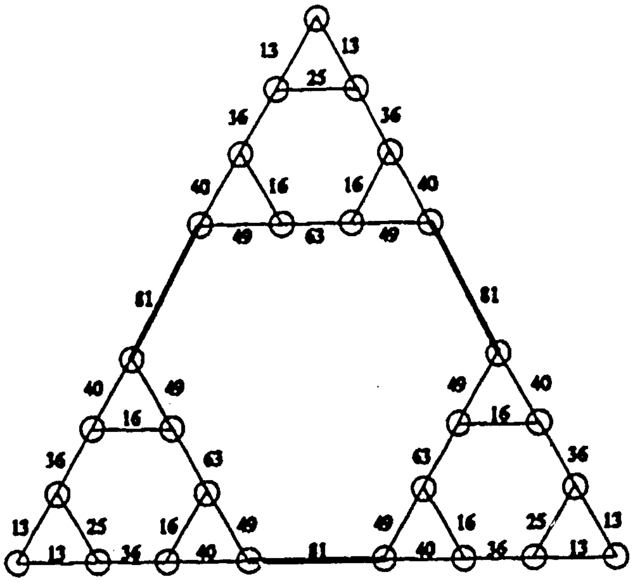

Figure 4.16: Packets transferred through each link if each node sends messages to all the other nodes in a (3, 3)-WKR.

$$\alpha = \frac{W^{L-1} \times W^{L-1}}{N \times N \times D_{avg}} = \frac{W^{2L-2}}{N^2 D_{avg}} \tag{4.53}$$

where $N = W^L$ is the total number of nodes. To find the average distance a message travels in a WK-Recursive network under uniform random destination of messages, we find a closed form expression for the recursive equation presented in [21] which yields the average message distance. If $D(W, L)$ represent the diameter of a (W, L)-WKR, $H(W, L)$ be the average distance between any two nodes, $H'(W, L)$ be the average distance between two nodes that lie in two distinct $(W, L-1)$-WKR clusters, and $H''(W, L)$ be the average distance between two nodes of a (W, L)-WKR such that one node is the corner node of the (W, L)-WKR, the following equations hold

$$\begin{aligned} H''(W, L) &= \frac{(W-1)(H''(W, L-1) + D(W, L-1) + 1) + H''(W, L-1)}{W} \\ H'(W, L) &= 2H''(W, L-1) + 1 \end{aligned}$$

$$H(W,L) = \frac{H(W,L-1) + (W-1)H'(W,L)}{W}$$

Solving the above difference equations we derive the closed form solution for each parameter

$$H''(W,L) = \left(\frac{W-1}{W}\right)(2^L - 1) \text{ for } L \geq 0 \tag{4.54}$$

$$H'(W,L) = \left(\frac{W-1}{W}\right)2^L + \frac{2-W}{W} \text{ for } L > 0 \tag{4.55}$$

$$H(W,L) = \left(\frac{2(W-1)^2}{W(2W-1)}\right)2^L + \left(\frac{1}{1-2W}\right)\left(\frac{1}{W}\right)^L + \frac{2-W}{W} \tag{4.56}$$

In this section we find a closed-form expression for the waiting delay a packet incurs in a link between the two $(W,L-1)$-WKR clusters of a (W,L)-WKR network. Under random destination of messages this waiting time is the worst among all the links and is a good representation of the network condition. In section 4.3 we study the effect of communication locality on this delay. Again we assume that a queue of unbounded capacity is associated with each output port and in each cycle a unit-sizea packet leaves the queue. This causes the probability of a packet arriving at an incoming channel be the same as the channel utilization, ρ. If the probability of a message injection by a node in a cycle is λ_{inj}, the probability of a message arriving at a specific channel connecting two $(W,L-1)$-WKR clusters is

$$\rho = \lambda_{inj} N D_{avg} \alpha = \lambda_{inj} W^{L-2} \tag{4.57}$$

Figure 4.17 shows the model of a node switch. The packet probability, ρ, in the channel of interest is composed of k (or W) components, ρ_i, ρ_1, upto ρ_{k-1}. ρ_i corresponds to the packet injected into the channel by the processor or received from

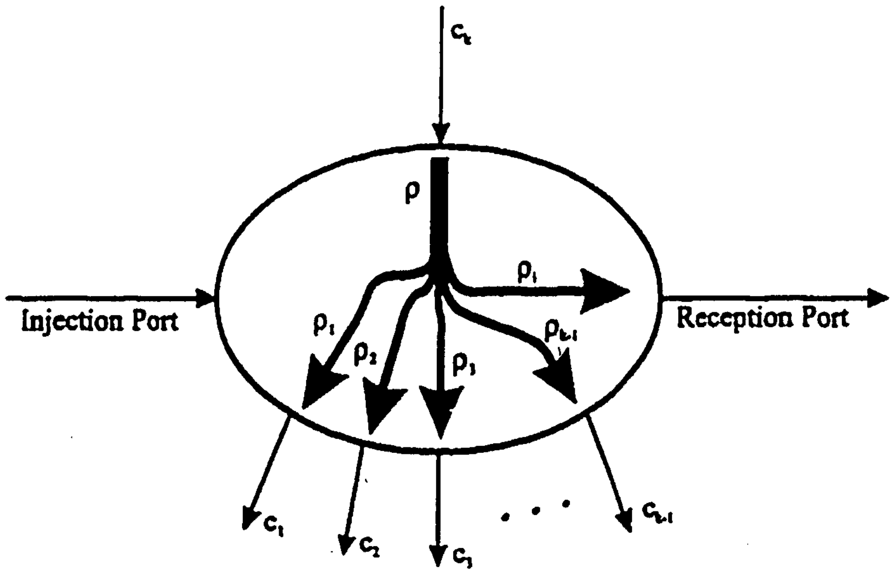

Figure 4.17: Switching probabilities of the channel.

the channel by the processor. As before, we assume that the injected packets at a node are directed to output ports with equal probability; therefore, $\rho_i = \lambda_{inj}/W = \rho/W^{L-1}$. Also, ρ_1 through ρ_{k-1} correspond to the packets steered into the remaining channels on the node from the channel of interest. We can see from Figure 4.16 that the ratio of the messages passed through each of the channels to total messages passed through all channels is

$$\begin{aligned} \beta &= \frac{W^{L-1} \times (1 + W + \ldots + W^{L-2})}{N \times N \times D_{avg}} \\ &= \frac{W^{L-1} \times \left(\frac{W^{L-1}-1}{W-1}\right)}{W^{L} \times N \times D_{avg}} \\ &= \frac{W^{L-1} - 1}{NW(W-1)D_{avg}} \end{aligned}$$

Therefore, the probabilities of a message arriving at each of the remaining channels on the node are equal, called ρ_c, and is

$$\rho_c = \lambda_{inj} N D_{avg} \beta = \left(\frac{W^{L-1} - 1}{W(W-1)} \right) \lambda_{inj} \tag{4.58}$$

Substituting for λ_{inj} from equation 4.57 yields

$$\rho_c = \frac{\rho(1 - W^{1-L})}{(W-1)} \tag{4.59}$$

Using the above information, we write the distribution for $\tilde{\nu}$ as

$$p(\tilde{\nu}) = \begin{cases} (1-\rho_i)(1-\rho_c)^{k-1} & \tilde{\nu} = 0 \\ \rho_i(1-\rho_c)^{k-1} + (k-1)(1-\rho_i)\rho_c(1-\rho_c)^{k-2} & \tilde{\nu} = 1 \\ (k-1)\rho_i\rho_c(1-\rho_c)^{k-2} + \binom{k-1}{2}(1-\rho_i)\rho_c^2(1-\rho_c)^{k-3} & \tilde{\nu} = 2 \\ & \vdots \\ \binom{k-1}{j-1}\rho_i\rho_c^{j-1}(1-\rho_c)^{k-j} + \binom{k-1}{j}(1-\rho_i)\rho_c^j(1-\rho_c)^{k-j-1} & \tilde{\nu} = j \\ & \vdots \\ \rho_i\rho_c^{k-1} & \tilde{\nu} = k \\ 0 & \tilde{\nu} > k \end{cases} \tag{4.60}$$

We find the expectation of this distribution which will be

$$\begin{aligned} E[\tilde{\nu}] &= \sum_{\tilde{\nu}=1}^{k} \tilde{\nu} p(\tilde{\nu}) \\ &= \rho_i + (k-1)\rho_c \\ &= \rho \end{aligned} \tag{4.61}$$

and the variance of the distribution will be

$$\begin{aligned} Var[\tilde{\nu}] &= E[\tilde{\nu}^2] - E[\tilde{\nu}]^2 \\ &= \sum_{\tilde{\nu}=1}^{k} \tilde{\nu}^2 p(\tilde{\nu}) - \rho^2 \\ &= \sum_{\tilde{\nu}=1}^{k} \tilde{\nu}^2 \left[\binom{k-1}{\tilde{\nu}-1} \rho_i \rho_c^{\tilde{\nu}-1} (1-\rho_c)^{k-\tilde{\nu}} + \binom{k-1}{\tilde{\nu}} (1-\rho_i) \rho_c^{\tilde{\nu}} (1-\rho_c)^{k-\tilde{\nu}-1} \right] - \rho^2 \end{aligned} \tag{4.62}$$

4.2 Routing

Under dimension-order routing, packets from different input channels at a node are routed to an output channel with different probabilities. This effect can be mitigated if adaptive routing is employed. Since, under adaptive routing, the packets do not traverse the dimensions in a specific order, we can assume that packets route to each dimension at random. This causes the packet arrivals to an output queue to have a binomial distribution.

Considering the k-ary n-cube, in one cycle, an output queue in a dimension can accept upto $n-1$ packets from the input channels in the remaining dimensions and one packet from the injection port for a total of n packets. At each cycle, a packet arrives at an output channel with the probability ρ where $\rho = \frac{\lambda_{inj} D_{avg}}{n}$. If we assume a minimal adaptive routing, under which a packet travels the same distance as in dimension-order-routing, $D_{avg} = nk_d$. Thus, the probability of packet arrivals to an output channel will be, $\rho = \lambda_{inj} k_d$. Under adaptive routing, all the input channels contribute equally to this traffic. Therefore, the probability that a packet is arrived at an output channel from any input channel, or the injection port, will be λ_{inj}.

If we again assume that $\tilde{\nu}$ is the number of packets joining a fixed output queue, we can see that $\tilde{\nu}$ has a simple Bernoulli distribution $b(.;n;\lambda_{inj}/n)$. The expected number of arrivals will be

$$E[\tilde{\nu}] = n(\lambda_{in_j}/n) = \lambda_{in_j} \qquad (4.63)$$

and the variance of $\tilde{\nu}$ will be

$$Var[\tilde{\nu}] = n(\lambda_{in_j}/n)(1 - \lambda_{in_j}/n) = \lambda_{in_j}(1 - \lambda_{in_j}/n) \qquad (4.64)$$

Plugging the above equations in 4.21, we get the average waiting time in a k-ary n-cube under minimal adaptive routing

$$\overline{w} = \frac{(1 - 1/n)\lambda_{in_j}}{2(1 - \lambda_{in_j})} \qquad (4.65)$$

Several adaptive routing algorithms have been proposed for k-ary n-cubes. Chalasani and Boppana compare a few of these algorithms in [8]. Appendix B of this thesis presents two adaptive routing algorithms for generalized hypercubes and WK-Recursive structures.

4.3 Communication Locality

In most of our analyses, we assumed that the message destinations were randomly chosen from all the nodes in the network. Despite several software practices such as memory interleaving and uniform distribution of parallel data structure which tend to spread accesses uniformly over all nodes, this type of distribution rarely happens in practice. Majority of software practices attempt to decrease the communication overhead by increasing the locality of message destinations. Communication locality improves the performance of the network by decreasing the base network latency and also limiting the network bandwidth required by the application.

One major advantage of direct networks over indirect networks is that they can take advantage of the locality in parallel applications. Informally, we say that communication locality exists when the likelihood of communication to various nodes decreases with distance. Packets destined for neighboring nodes not only travel fewer hops, but also consume a smaller fraction of the network bandwidth. In this section, we include the effect of communication locality on the latency models we developed previously.

We can extend our models to account for communication locality by introducing a simple locality model. We define the locality fraction ℓ as the fraction of all processors that are potential candidates to receive a message from a source node. Furthermore, for a given source node, we allow the message destinations be randomly chosen from a subdivision with a smaller diameter than the entire network. In k-ary n-cubes and GHCs this subdivision can be an n-dimensional subcube with $N \times \ell$ nodes centered at the source node. In a (W, L)-WKR, a (W, L')-WKR, where $L' < L$ and contains the source can bound the message destinations.

Focusing on the k-ary n-cubes, let us consider a two-dimensional N-processor torus in which nodes are represented by their x and y coordinates. Given a locality fraction ℓ, destination nodes for messages originating from source node (i, j) are randomly chosen from the set of nodes with coordinates $(x \mid i \leq x \leq i + \sqrt{\ell N} - 1, y \mid j \leq y \leq j + \sqrt{\ell N} - 1)$. Other forms of communication locality could also be realized by using some probability function to represent higher than average access likelihoods to nearby nodes, or to favor straight through paths over paths that require turns.

With the above locality model, a packet travels an average distance of k_{dl} in each dimension, for a total of nk_{dl} hops from source to destination. Under two-dimensional mapping, the average distance traversed in a dimension can be expressed as

$$k_{dl} = ((\ell N)^{1/n} - 1)/2 = (\ell^{1/n}k - 1)/2 \qquad (4.66)$$

The average latency can be derived by replacing k_d in equations 4.43 and 4.44 with k_{dl}. The same substitution is necessary in 4.42 to compute the channel utilization, ρ. Destinations chosen randomly over the entire machine correspond to $\ell = 1$.

Locality affects the network performance by decreasing the latency and increasing the effective throughput of the network. Ideally, the network reaches full capacity when all channels are fully utilized, that is, when $\rho = \frac{L\lambda_{inj}k_d}{W} = 1$. The peak network throughput is messages per cycle per node is $\frac{1}{Bk_d}$, where $B = \frac{L}{W}$; or is $\frac{1}{k_d}$ in flits per cycle per node. However, when communication locality exists, the throughput increases to $\frac{1}{k_{dl}}$ flits per cycle per node. Similarly, the base network latency of $nk_d + B$ hops under cut-through switching decreases to $nk_{dl} + B$ when locality exists. Under store-and-forward, the base network latency decreases from nk_dB to $nk_{dl}B$. In other words, locality increases throughput by a factor $\frac{1}{\ell^{\frac{1}{n}}}$, and decreases the base network latency by the same factor (under cut-through switching, when $nk_d \gg B$).

Locality improves latency because it reduces both the number of hops per packet and average contention delays. As displayed in Figures 4.18 and 4.19, with a light load of $\lambda_{inj} = 0.001$, latency reduction is largely due to the fewer number of hops. At light loads, latency is linearly related to k_{dl} or to $\ell^{1/n}$, which is clear from 4.43 and 4.44 when the contention component is ignored. For example, when $\lambda_{inj} = 0.001$, for a 1K-node machine ($n = 2$ and $k = 32$), the average latency for randomly selected

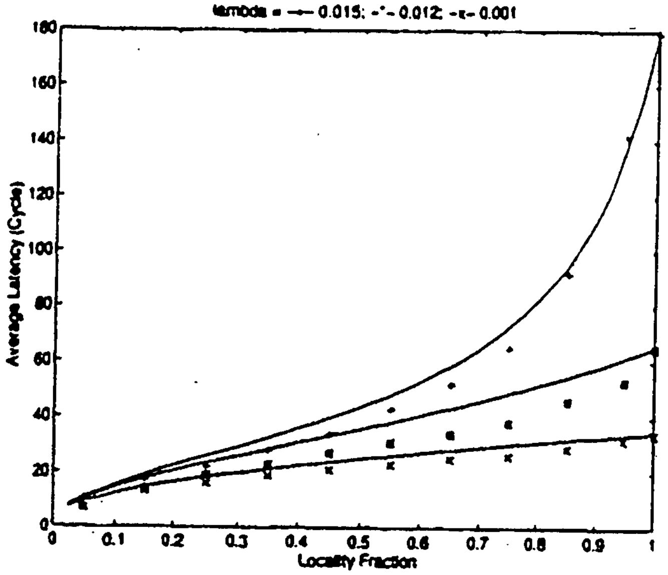

Figure 4.18: Effect of locality on communication bandwidth and latency on a k-ary n-cube with $n = 2$, $k = 32$, and $B = 4$ under cut-through switching. Dashed lines correspond to model predictions.

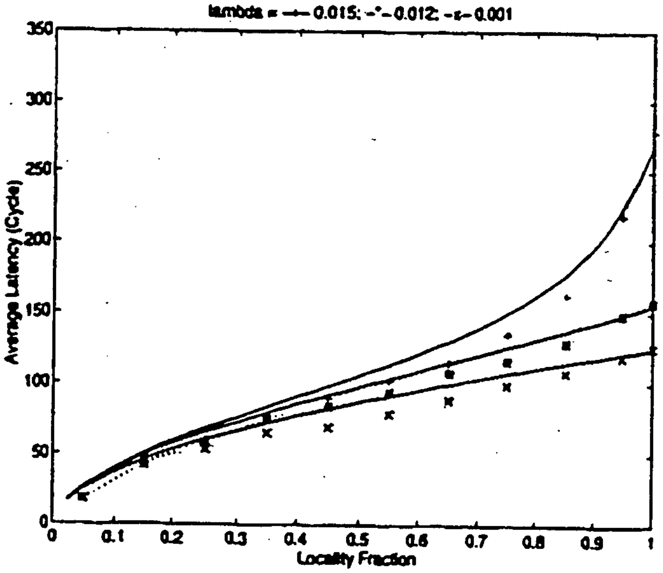

Figure 4.19: Effect of locality on communication bandwidth and latency on a k-ary n-cube with $n = 2$, $k = 32$, and $B = 4$ under store-and-forward switching. Dashed lines correspond to model predictions.

destinations is roughly 35. When the average distance in a dimension decreases by 10% ($\ell^{1/2} = 0.9$). the latency decreases by the same factor to 31.

The impact of locality is much more significant when contention is high. In this case. the latency reduction due to locality is largely due to a reduction in the bandwidth requirement. The latency at high loads is proportional to $\frac{1}{1-\lambda_{inj}Bk_d}$. For example. the average latency drops by over 25% (from 67 to 50) for the higher load of $\lambda_{inj} = 0.012$. when $\ell^{1/2} = 0.9$. Of this decrease. over 19% is due to the reduced bandwidth consumed. while less than 6% is due to the fewer number of hops. Thus. we see that communication locality has a much stronger effect on network performance through reducing the consumed bandwidth than through reducing the base network latency. The proportional impact of locality is even more significant at higher loads.

When communication locality exists. low-dimensional networks outperform networks with higher dimensions. Although low-dimensional networks have shorter wires and smaller bisections. their lower available bandwidth and higher base latencies reduce their effectiveness. Locality mitigates these negative aspects of low-dimensional networks by reducing the effective distance a message travels. consequently decreasing bandwidth requirements and the base latency.

CHAPTER 5
SINGLE-MODE TRAFFIC COMMUNICATION

Future multicomputers will run a broad group of applications which require distinct qualities of service from the multicomputer network. The coexistence of real-time and non-real-time applications in these systems necessitates a fresh look on the parameters of these networks and a reevaluation of their design criteria. The exhibited quality of services depend not only on the network properties. such as channel width. diameter. node delay. wire delay. switching technique. and routing, but on the load characteristics. such as the message inter-arrival time distribution. the message length. and the degrees of the communication locality. In the previous chapter. we investigated the effect of these architectural and load parameters on the performance of the network through analysis and simulation. In this chapter. we focus on the switching schemes and compare their average performance and predictability through simulations.

In a multicomputer network, the switching scheme controls the flow of packets through the network by exercising different resources at nodes along a packet's route. This section evaluates the ability of wormhole, virtual cut-through, and store-and-forward switching to accommodate different performance requirements. Each switching scheme is best-suited for certain traffic classes with particular characteristics and performance requirements. The method each switching scheme employs to allocate buffer and link resources determines the average packet latency and its variance and also affects the influence of in-transit packets on other network traffic.

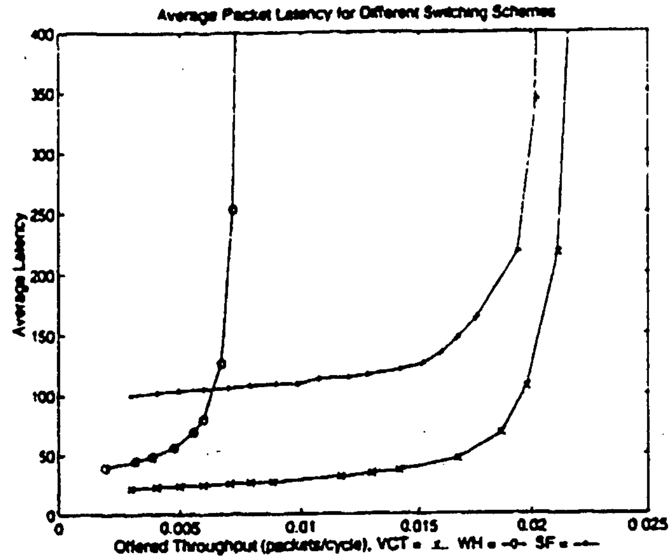

Figure 5.1: Average latency on a packet transfer.

5.1 Average Latency

Store-and-forward switching scheme requires an arriving packet to buffer completely before transmission to a subsequent node can begin. In contrast. cut-through methods. such as virtual cut-through [30] and its special case. wormhole switching, [12], try to forward an incoming packet directly to an idle output link. If the packet encounters a busy outgoing channel. virtual cut-through switching buffers the packet, while a blocked wormhole packet stalls in the network pending access to the link.

To perform a comparative study on the three switching schemes we ran a series of simulations on an 8 x 8 mesh network with one virtual channel per physical link and carrying 16-flit packets using dimension-ordered routing. Each node generates packets with exponentially-distributed interarrival times and uniform random selection of destination nodes. Figure 5.1 shows the average end-to-end packet latency for the three switching schemes as a function of average offered throughput per node which is the average number of packets generated in a single cycle by a node. As we can see in the figure. both wormhole and virtual cut-through switching perform well at low loads by avoiding unnecessary packet buffering at intermediate nodes: however. wormhole performance degrades abruptly with an increase in traffic. At high loads. virtual cut-through and store-and-forward performance gradually merge. as high network utilization decreases the likelihood that an in-transit packet encounters an idle output link.

Virtual cut-through and store-and-forward consume network bandwidth proportional to the offered load by removing blocked packets from the network. On the other hand. a blocked wormhole packet stalls in the network. effectively dilating its length until its outgoing channel becomes available. As a result. wormhole networks typically utilize only a fraction of the available network bandwidth [15. 38]. as seen by the early saturation of the wormhole plot in Figure 5.1. At higher loads. this effect enables store-and-forward to outperform wormhole switching, even though store-and-forward introduces buffering delay at each hop in a packet's route. As we will see later. if virtual channels are used in conjunction with wormhole switching. the effect of blocking can be mitigated to some extent: however. channel contention still creates dependencies amongst packets spanning multiple nodes.

The sensitivity of wormhole networks to slight changes in load. including short communication bursts [16], complicates the use of wormhole switching for guaranteed traffic. We will delve into this effect later. In spite of this sensitivity, wormhole switching is particularly well-suited to best-effort packets, due to its low latency and minimal buffer space requirements. While flow-control costs limit the utility of wormhole switching in distributed systems, parallel machines and multicomputer networks can dynamically transfer or stall wormhole flits without complicating buffer allocation for other traffic. In the next chapter, we will describe how, with effective flow-control

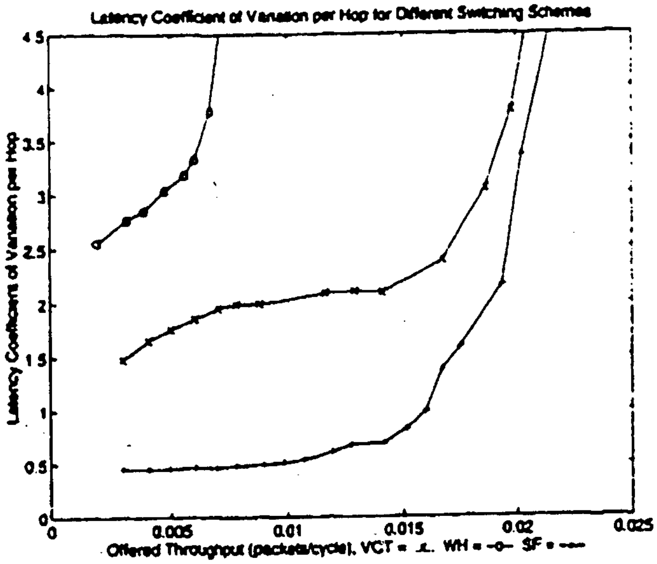

Figure 5.2: Latency coefficient of variation on a hop.

and arbitration schemes. best-effort packets can employ wormhole switching without compromising the performance of the guaranteed traffic.

5.2 Predictability

Although best-effort communication requires low average latency. guaranteed communication demands predictable network delay and throughput. A good measure of predictability is the coefficient of variation which is the ratio of the standard deviation to the mean [34]. Figure 5.2 shows the coefficient of variation for packet latency for the three switching schemes. Since latency characteristics vary depending on the distance between source-destination pairs, the graphs present the changes in coefficient of variation for latency per hop.

Across all loads, store-and-forward incurs the least variability since packets deterministically buffer at intermediate nodes. Coupled with static routing, a store-and-forward transfer utilizes deterministic buffer and channel resources at fixed nodes and links along the route. This greatly simplifies the allocation and scheduling of

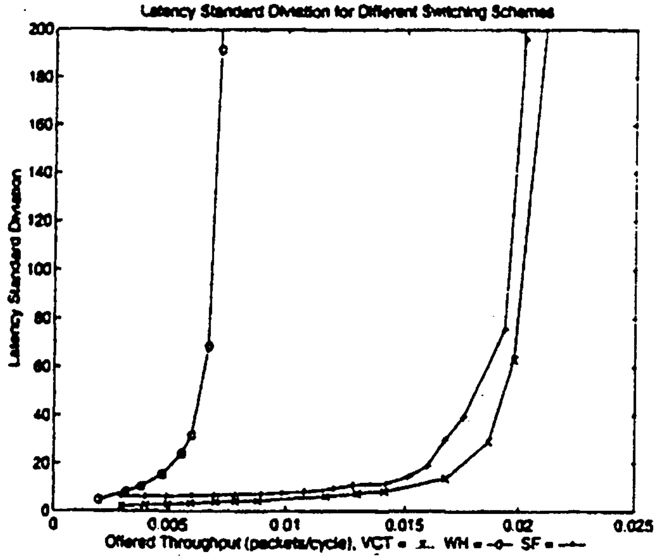

Figure 5.3: Latency standard diviation on a hop.

resources throughout the interconnection network. In contrast, virtual cut-through imparts variable load on memory resources at intermediate nodes by basing the buffering decision on the status of the output links. At high loads, virtual cut-through and store-and-forward merge, as in Figure 5.1, due to the decreasing likelihood of packet cut-throughs.

On the other hand, in wormhole switching a packet is never buffered and consumes unpredictable amounts of channel bandwidth by stalling in the network. In Figure 5.2, wormhole latency variation increases dramatically with rising load, even under a moderate injection rate below saturation throughput. Below the saturation load, wormhole switching results in a low average latency, as seen in Figure 5.1, but a portion of the traffic incurs larger delay due to pockets of channel contention. In addition to a large coefficient of variation, wormhole traffic suffers a large standard deviation of packet latency, as shown in Figure 5.3.

CHAPTER 6
SUPPORT FOR MULTIPLE CLASSES OF TRAFFIC

Best-effort and guaranteed traffic have conflicting performance goals that complicate interconnection network design. The effective mixing of guaranteed and best-effort traffic hinges on controlling the interaction between these two classes. In particular. best-effort packets cannot consume arbitrary amounts of link or buffer resources while guaranteed packets await service.

6.1 Architecture

As seen in Chapter 5. wormhole and packet switching exercise complementary resources in the interconnection network. with wormhole switching reserving virtual channels and packet switching consuming buffers in the router. Hence. the combination of wormhole switching for best-effort traffic and packet switching for guaranteed communication enables effective partitioning of router resources. However. since the traffic classes share network bandwidth. the router must regulate access to the physical links to control the interaction between the two classes.

Assigning the best-effort and guaranteed packets to separate virtual networks can regulate this interaction between the traffic classes. The router divides each physical link into multiple virtual channels, where some virtual channels carry best-effort packets and the rest accept only guaranteed traffic. Virtual channels provide an effective mechanism for reducing the interaction between packets while still allowing traffic to share network bandwidth. Several router architectures utilize virtual

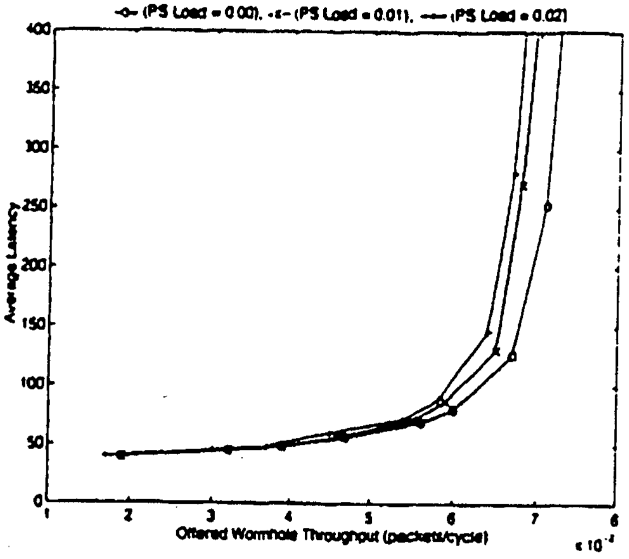

Figure 6.1: Effect of packet switching load on wormhole average latency.

channels to separate packets into classes, such as control and data, where the classes employ the same routing and switching scheme [17]. Exporting the virtual channel abstraction to the injection and reception ports further prevents intrusion between packets at the network entry and exit points.

By tailoring the routing, switching, and flow-control policies for each virtual network, multicomputer routers can support traffic classes with conflicting performance requirements. Packets on separate virtual networks interact only to compete for access to the physical links and ports. This bounds network access time for guaranteed packets, independent of the amount or length of best-effort packets. The communication software, or hardware, can then build on this underlying abstraction to provide various services, such as connection-oriented communication with latency or bandwidth guarantees. Fine-grain flow control on the wormhole virtual network enables best-effort flits to capitalize on slack link bandwidth left unclaimed by guaranteed packets.

6.2 Arbitration

Figures 6.1. 6.2. and 6.3 evaluate the effect of increasing best-effort load on the performance of both best-effort and guaranteed traffic in this router architecture. In these experiments. the router interleaves two virtual channels on each link. with one virtual channel allocated to best-effort packets for wormhole routing and one dedicated to guaranteed traffic using store-and-forward. Each curve shows the impact of changing best-effort load in the presence of a fixed rate of injection for guaranteed packets. The router employs round-robin arbitration amongst the active virtual channels contending for each link.

Figure 6.1 shows the average latency for the best-effort. wormhole packets. under three different injection rates for the store-and-forward or packet-switched (PS) traffic. Note that the curve for zero packet-switched load corresponds to the wormhole latency data in Figure 5.1. As the amount of wormhole traffic increases. best-effort packets incur larger latency due to increased channel contention within the best-effort virtual network. Even with fairly heavy packet-switching load. the best-effort packets maintain low average latency until reaching the saturation throughput.

The presence of packet-switched traffic does not significantly limit this achievable best-effort throughput, since the wormhole virtual network saturates due to virtual channel contention, not a shortage of network bandwidth.

As seen in Figures 6.2 and 6.3, both the average latency and predictability of the guaranteed packets are largely unaffected by the best-effort traffic, due to fine-grain arbitration amongst the virtual channels. For both packet-switched loads, the mean and standard deviation of end-to-end latency closely match the corresponding values in Figures 5.1 and 5.3, even as the wormhole traffic exceeds its sustainable load. Channel contention on the best-effort virtual network does not impede the

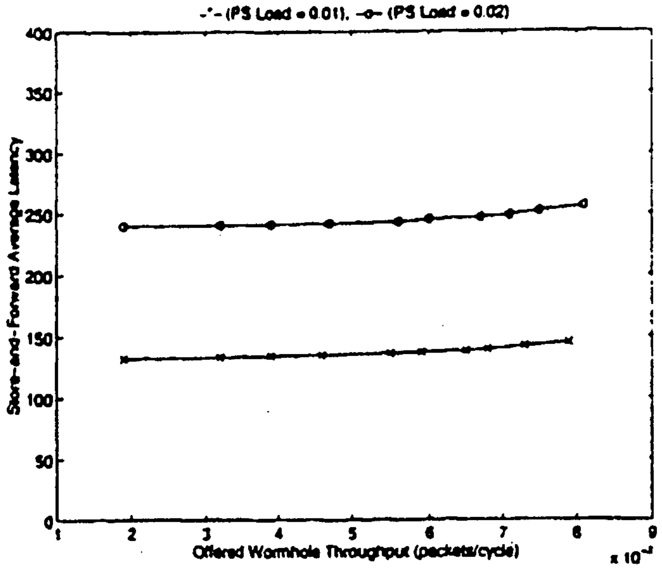

Figure 6.2: Effect of wormhole load on store-and-forward average latency.

forward progress of guaranteed packets, since blocked wormhole packets temporarily stall in their own virtual network instead of depleting physical link or buffer resources. Demand-driven arbitration ensures that either class of traffic can improve throughput by capitalizing on the available link bandwidth.

While the separate virtual networks limit the interaction between the traffic classes, the arbitration for access to the physical link still permits active best-effort virtual channels to increase delay for the guaranteed packets. This is manifested in Figures 6.2 and 6.3 by the slight increase in packet-switching latency and standard deviation in the presence of a heavier load of wormhole traffic. More significantly for the guaranteed traffic, fair arbitration amongst the virtual channels varies the service rate afforded both traffic classes, providing slower guaranteed service under increasing best-effort load.

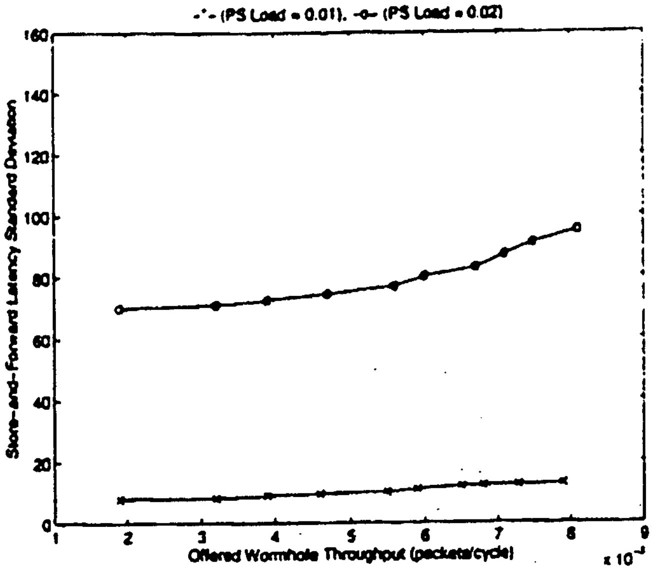

Figure 6.3: Effect of wormhole load on store-and-forward latency standard deviation.

6.3 Control of the Guaranteed Traffic

The router can further minimize intrusion on guaranteed traffic by imposing priority arbitration between the virtual networks, where guaranteed packets always win arbitration over the best-effort packets. For a guaranteed packet, this effectively provides flit-level preemption of best-effort traffic across its entire path through the network. Unlike the results in Figures 6.2 and 6.3, assigning priority to guaranteed traffic removes any sensitivity to the best-effort load. Priority arbitration enables a guaranteed packet to travel at the same rate through each link in its journey, independent of the number of active best-effort virtual channels. This abstraction enables the scheduler to allocate resources based only on the worst-case requirements of the guaranteed traffic, while still enabling best-effort traffic to dynamically consume unused link bandwidth.

However, priority arbitration can exact a heavy toll on the best-effort packets, particularly at higher loads, as illustrated by Figure 6.4. This graph shows the average

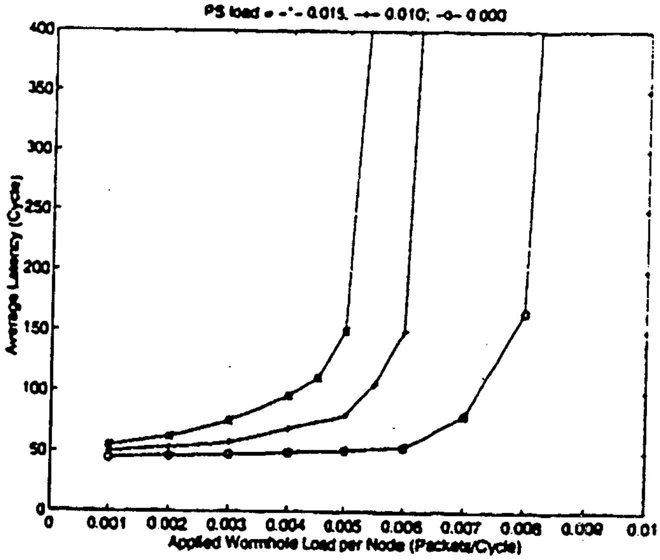

Figure 6.4: Average wormhole latency using a priority-based arbitration scheme.

latency of best-effort wormhole packets in the presence of three different packet-switching loads under priority arbitration for the physical links. Unlike Figure 6.1. Figure 6.4 shows significant degradation of the performance of best-effort packets, since the strict priority-based scheme restricts their forward progress. Even in the absence of livelock, lengthy blocking of wormhole flits increases contention delays in the best-effort virtual network.

Priority arbitration varies the service rate for the best-effort packets depending on the load of guaranteed traffic. To reduce contention, the best-effort virtual networks could employ adaptive routing to enable these packets to circumvent links and nodes serving a heavy load of guaranteed packets. Alternatively, the router could aid the forward progress of best-effort packets by ensuring predictable access to the physical link, even in the presence of guaranteed packets. The router can allow up to α best-effort flits to accompany the transmission of a guaranteed packet. Since the guaranteed traffic employs packet switching, a guaranteed packet holds the physical link for a bounded time proportional to its packet length ℓ. In effect, this dilates each guaranteed packet to a service time of at most $\ell + \alpha$ cycles, while dissipating contention in the best-effort virtual network. When no guaranteed packets await service, pending best-effort flits have free access to the outgoing link.

This permits forward progress for best-effort. packets while still enforcing a tight bound on the intrusion on guaranteed traffic, without restricting packet size. Such a credit-based scheme preserves necessary delay abstractions for the scheduling of guaranteed traffic. For additional flexibility, a writeable register in each router would allow the system to set α when downloading tasks to the processing nodes. For example, the compiler could test the schedulability of the guaranteed communication under several candidate α values, selecting an α that does not disrupt the delay or bandwidth bounds for the guaranteed packets. This enables the compiler to determine the appropriate trade-off between the best- effort performance and the admission of guaranteed traffic for a given application.

CHAPTER 7
CONCLUSION

In this chapter we review the contributions of this dissertation. and allude to the possible extensions and future research for the work presented.

7.1 Research Contributions

In this dissertation. we examined closely the assumptions and requirements of multicomputer network design and reevaluated their parameters to see how they could deliver the diverse performances required by modern applications. We investigated how the conflicting performance goals of best-effort and guaranteed traffic affect the suitability of routing, switching, and flow control schemes in the network.

As part of the work. we created a general evaluation framework which allowed us to characterize the performance of multicomputer networks as a wide range of parameters are varied. We developed a simulation environment. called RSIM. which is programmable to the network topology, the network size, the routing algorithm. the switching scheme, the number of virtual channels, the allocation of virtual channels to subnetworks, the message generation distribution, the message destination distribution, the message length, and the types of evaluation metrics. The data collected from the simulator were used to test the developed models and also served as the primary source whenever it was difficult to derive accurate analytical models.

We modeled the latency in k-ary n-cubes, generalized hypercubes, and WK-Recursive networks, under cut-through and store-and-forward switching schemes, with or without contention. The network analysis under no contention presented the base network latency and allowed us to analyze the relative effect of wire and switch delays under various constraints such as fixed wire bisection width. fixed channel widths. and fixed node sizes. We employed constant. linear. and logarithmic wire delay models in our analysis. We also developed closed-form expressions for contention in buffered direct networks to estimate the effect of network bandwidth. We validated the models through simulations and demonstrated their robustness over a wide range of network sizes. The contention models were merged with the base network results to obtain the complete latency models for the multicomputer networks.

An interesting finding of this analysis was that a relative standing of networks was strongly dependent on the constrains chosen and on the expected workload parameters. In contrast. the results showed much less variance when bandwidth considerations were ignored.

We also investigated the effect of communication locality on the performance of the direct networks. Unlike indirect networks. locality improves both network throughput and latency. At low loads. communication latency decreases linearly with the average distance a packet traverses. Under high network load. locality has even more effect on the latency, mainly due to the reduction in the bandwidth required by the application. We observed that the effect of communication locality is more obvious on networks with low number of dimensions. Although these networks have shorter wires and smaller bisection widths than other networks, their lower available bandwidth and higher base latencies reduce their effectiveness. Locality mitigates these negative aspects of low-dimensional networks by reducing the average distance a message travels. The decrease in the message distance reduces both the network base latency and the bandwidth requirements of the application.

We studied the effect of adaptive routing on the network latency. We modified the developed network latency models to address this type of routing. Adaptive routing distributes the load more evenly across the network and results in a better network bandwidth. The results obtained from the model agreed with this statement.

Next, we examined how different switching schemes satisfy different latency and predictability requirements. We generated simulation results for average latency, latency standard deviation, and latency coefficient of variation for each switching type. We observed that both virtual cut-through and wormhole switching performed well at low load by avoiding unnecessary packet buffering at intermediate nodes: however wormhole performance degraded abruptly with increasing traffic. The performance of virtual cut-through and store-and-forward switchings merged at high loads.

From a predictability point-of-view, store-and-forward incurred the least variability in latency, across all loads, mainly due to its deterministic buffering scheme at intermediate nodes. In contrast virtual cut-through and wormhole switching impart variable amount of load on the memory or channels of intermediate nodes, respectively, they show large variation in latency. The predictability of store-and-forward switching makes it suitable for real-time applications which require a guaranteed performance.

Finally, the dissertation establishes a paradigm for the efficient and reliable mixing of guaranteed and best-effort traffic in message-passing multiprocessors. Unlike the previous work in this area which has mostly been focused on the software protocol schemes, we propose architectural features which exercise efficient, fine-grain control over the interaction of packets. In order to optimize for the performance requirements of each class, the architecture employs different routing and switching strategies to manage the two traffic classes. We provide tight bounds on the intrusion of best-effort traffic on guaranteed packets by low-level control of the network access time and bandwidth allocation. The software or the higher level hardware can utilize these bounds to provide the quality of service required by the application.

7.2 Future Directions

In our simulations, we used synthetic loads because they are simple to produce and they are storage-efficient as they can be produced using pseudo-random number generators. The primary deficiency of synthetic workloads is their lack of communication structure and message correlation that is typical in actual parallel applications. For example, our workloads did not couple the initiation of messages with the completion of other messages, a typical program feature.

In the future, we should study the interconnection networks under natural loads, as well. We can use traces of captured message traffic, or even actual message traffic. It is also possible to develop a machine with a programmable interconnect that actually run the application with different network configurations to evaluate the architecture of the network interconnection in real-time with an application program.

An important point, which can make the simulations easier and more accurate is to provide a better method of detecting when the system has reached the steady state. currently, we inject a fixed number of packets before starting the data collection, just to bring the system into the steady state. Determination of this number can be a tricky and difficult task. For example, throughput beyond saturation load can depend on the actual length of the simulation that was run, due to residual traffic in the source queues. Such dependencies should be avoided, if possible, but reporting simulation run lengths would at least make such figures detectable.

As a future extension to this project. we should develop a router architecture prototype based on the model presented in this dissertation. The router should initially be designed using a hardware definition language such as VHDL. completely simulated. and finally be prototyped. The router will be programmable to different routing and switching schemes and can execute these tasks at the fine level. reducing the overhead incurred in the software messaging layer.

APPENDIX A
THE RSIM SIMULATOR

RSIM is an object-oriented simulation environment for evaluating multicomputer networks. The planned features for RSIM are many. but currently only the most general have been coded. The program is written as a module that may be used within a wide variety of other systems.

RSIM contains a user interface that reads commands from a file and runs the simulation thus defined. The command file is tokenized by a flex program. and parsed by a yacc program. The basic sections of the command file for RSIM are as follows:

RSIM contains a topology section in which the topology of the network is defined. The selection can be from a group of predefined topologies - mesh. torus. hypercube. and *WK*-Recursive, or it can be defined as a general topology.

TOPOPLOGY = {*Mesh. Hypercube. Torus. General*};

If a predefined topology is selected. two other fields in the configuration file define the dimension and the indices of the structure. If the *GENERAL* option is used, the complete connectivity has to be specified. In the connectivity section of the command file, each node has a unique ID number - connections to a node are specified by associating a source node ID with a destination ID. All of the connections specified are assumed to be uni-directional, as bi-direction channels can be effectively simulated by two opposing uni-directional ones. With this method virtually any topology is possible, but the user must meticulously write in the command file (at least) the destination ID of every connection. This is unacceptable for systems that have a very large number of nodes (hundreds or perhaps thousands), and the user is encouraged to take advantage of the predefined topologies.

The network architectural and load parameters are completely specified in this file. The format for these parameters is as follows:

NUMBER_OF_VIRT_CHANS = :

CHAN_WIDTH = {channel width}:

SEND_Q_DEPTH = :

RECV_Q_DEPTH = :

FLIT_SIZE = :

PACKET_SIZE = :

HEADER_SIZE = :

PEND_PKT_BUF_SZ = :

D = *dimension* {for non-General}, degree {for WK-Recursive}:

$K = k_{d-1}, k_{d-2}, \ldots, k_1, k_0$ {For non-General and non WK-Recursive};

CHAN_Q_SIZE = $q_{d-1}, q_{d-2}, \ldots, q_1, q_0$ {For non-General and non WK-Recursive};

Connectivity Table: {if Topology=*General*}

OF NODES =

Node: Node, Node, ...;

Node: Node, Node, ...;

Node: Node, Node, ...;

Node: Node, Node, ...;

Node: Node, Node, ...;

⋮

Transient and permanent failures can also be easily simulated by RSIM.

```
FAILED NODES:
node: (start time. end time).(start time. end time)....:
node: (start time. end time)....:
node: (start time. end time)....:
⋮
FAILED LINKS:
source node-dest node: (start time. end time).(start time. end time)....:
source node-dest node:(start time. end time)....:
 ⋮
```

The simulator is capable of running multiple loads simultaneously. These loads can each use different routing algorithms or switching schemes. Each load can also be generated based on a different generation distribution with different parameters. The provided distributions are *Exponential Uniform Random* and *Uniform Random* with user specified parameters. The provided routing algorithms are various oblivious and adaptive routing routines for each topology. User-created routing routines may also be easily used if it is coded into RSIM and compiled.

The simulation time is made up of *clicks*, analogous to clock ticks. during which each node is updated by one granular of time. A *step* is a collection of ticks over which the load parameters are held constant. At the beginning of each step the system load is increased by an amount defined in the command file. The initial load and number of ticks per step are similarly defined.

Each packet is allocated a unique id when it is sent. so that it may be identified when it is received. While the packet is in transit (ie: between creation and reception) information such as source node. destination node. and time initiated are kept. When the packet is received. this fact is reported to the data collection module. using the same packet id that was provided by the data collection module when its creation was reported. The transit information is added to running step totals (including the transit time histogram) and subsequently forgotten. When the simulation is complete. sim will report this data for each step.

The remaining fields of the configuration files are as follows:

```
NUMBER_OF_LOADS = :
LOAD = {UNIFORM_RANDOM. EXP_UNIFORM_RANDOM. ...}:
LOAD_CONTROL = {
        TASK_CONTROL 0 = {
                ROUTING_ALG = {DIM_ORDER_MESH. ...}:
                SWITCHING = {VCT. PACKET};
                NUMBER_OF_PACKETS_PER_STEP = :
                IGNORE_PACKETS = :
                IGNORE_TICKS = ;
                INIT_INJECT_RATE = :
                INJECT_RATE_INC = ;

                HIST_TICK_BASE = ;
                HIST_TICK_BLOCK_WIDTH = ;
                HIST_TICK_NUMBER_OF_BLOCKS = ;
                HIST_HOP_BASE = :
                HIST_HOP_BLOCK_WIDTH = :
                HIST_HOP_NUMBER_OF_BLOCKS = :
                HIST_TPH_BASE = :
                HIST_TPH_BLOCK_WIDTH = :
                HIST_TPH_NUMBER_OF_BLOCKS = :
                HIST_JOB_BASE = :
                HIST_JOB_BLOCK_WIDTH = :
                HIST_JOB_NUMBER_OF_BLOCKS = :
                HIST_CQO_BASE = :
                HIST_CQO_BLOCK_WIDTH = :
                HIST_CQO_NUMBER_OF_BLOCKS = :
                HIST_DLN_BASE = :
                HIST_DLN_BLOCK_WIDTH = :
                HIST_DLN_NUMBER_OF_BLOCKS = :
        };
        TASK_CONTROL 1 = {
                ⋮
};
STOPING_CRITERIA = {STEP_COUNT. ...};
NUMBER_OF_STEPS = :

END;
```

EXAMPLE As an example we show the configuration file for a 3 × 3 mesh with four virtual channels per physical channels. 32-bit channel widths. and eight-flit channel queue sizes for both virtual channels. The send and receive buffer sizes are both 512 bits. The flit size is 32 bits. The packet size and the packet header size are 8 and 1 flits. respectively. Finally, the pending packet buffer or the source buffer size is 16 packets.

We run only a single Uniform Random traffic on this network. The initial average injection rate per node is 0.125 packets/cycle. We increase this rate by 0.0625 packets/cycle after every step. In every step. we ignore the first 64 packets or 128 cycles (which ever is larger) and then collect 320 packets. We use dimension-order-routing and virtual cut-through switching. The simulation will run for five steps.

```
TOPOLOGY = MESH;
NUMBER_OF_VIRT_CHANS = 4;
CHAN_WIDTH = 32;
SEND_Q_DEPTH = 512;
RECV_Q_DEPTH = 512;
FLIT_SIZE = 32;
PACKET_SIZE = 8;
HEADER_SIZE = 1;
PEND_PKT_BUF_SZ = 16;
D = 2;
K = 3,3;
CHAN_Q_SIZE = 8,8;
```

```
NUMBER_OF_LOADS = 1;
LOAD = UNIFORM_RANDOM;
LOAD_CONTROL = {
        TASK_CONTROL 0 = {
                ROUTING_ALG = DIM_ORDER_MESH;
                SWITCHING = VCT;
                NUMBER_OF_PACKETS_PER_STEP = 320;
                IGNORE_PACKETS = 64;
                IGNORE_TICKS = 128;
                INIT_INJECT_RATE = 0.125;
                INJECT_RATE_INC = 0.0625;

                HIST_TICK_BASE = 0;
                HIST_TICK_BLOCK_WIDTH = 2;
                HIST_TICK_NUMBER_OF_BLOCKS = 50;
                HIST_HOP_BASE = 0;
                HIST_HOP_BLOCK_WIDTH = 1;
                HIST_HOP_NUMBER_OF_BLOCKS = 5;
                HIST_TPH_BASE = 1;
                HIST_TPH_BLOCK_WIDTH = 1;
                HIST_TPH_NUMBER_OF_BLOCKS = 5;
                HIST_JOB_BASE = 0;
                HIST_JOB_BLOCK_WIDTH = 1;
        HIST_JOB_NUMBER_OF_BLOCKS = 5;
        HIST_CQO_BASE = 0;
        HIST_CQO_BLOCK_WIDTH = 5;
        HIST_CQO_NUMBER_OF_BLOCKS = 20;
        HIST_DLN_BASE = -100;
        HIST_DLN_BLOCK_WIDTH = 10;
        HIST_DLN_NUMBER_OF_BLOCKS = 20;
    };
};
STOPING_CRITERIA = STEP_COUNT;
NUMBER_OF_STEPS = 5;

END;
```

APPENDIX B
ADAPTIVE ROUTING ALGORITHMS

To improve the network performance of highly parallel machines. the routing mechanism has to be able to diffuse the local congestion by adaptively utilizing the available resources in the network. In contrast with the deterministic routing in which the message trajectories are unique. in an adaptive routing scheme. they are continuously perturbed based on the condition of the network. In other words. packets are detoured to other available paths as local congestion or failures occur in the network. Adaptive routing will eliminate hot-spots in the network traffic by distributing the load throughout the entire network. Furthermore. by taking advantage of the inherent path redundancy in the richly-connected multicomputers. adaptive routing enhances the reliability of the system.

In this section. we present two adaptive routing algorithms. The first routine. GHC-P, is a progressive routing algorithm for generalized hypercubes. The latter is an adaptive routine for WK-Recursive structures.

B.1 Adaptive Routing in Generalized Hypercubes

In this section, we will develop an adaptive algorithm to route a message from one node to another in a generalized hypercube. In order to make the algorithm more effective in routing via the shortest path, the coordinates of the GHC are ordered in an increasing order, from right to left. Consequently, the rightmost coordinate will have the smallest modulus or the lowest number of nodes, and the leftmost coordinate

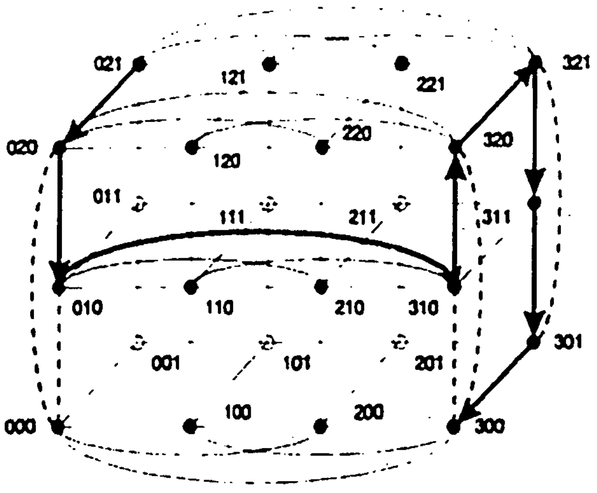

Figure B.1: An example of routing using GHC-P algorithm on a 4 × 3 × 2 GHC.

will have the largest modulus or the highest number of nodes. The routing is carried out. from right to left in the coordinates that are different in source and destination addresses. Following this technique. when the message gets closer to the destination. there will be more alternate paths in the dimension that the message is going through.

Since a GHC node may have more than one link in every dimension. a link at a specific node cannot be represented merely by the dimension it is located at. Two values are required to represent a link at a specific node. The first value represents the coordinate or dimension in which the link is located at. and the second value indicates the node to which the link is connected. For example. the link connecting the two nodes 001 and 002 is represented at node 001 by (1,2), and at node 002 by (1,1).

A path in a GHC can be represented by the source node and a *pathlist* which contains the links that the message has to traverse at consecutive nodes. For example, the path from node 021 to 300 of the GHC shown in Figure 2.1, can be represented by the source 021 and the list [(3,3), (1,0),(2,0)].

Algorithm GHC-P This progressive algorithm requires every node in the generalized hypercube to be aware only of the condition of its own links. The algorithm is able to route messages between any pair of non-faulty, or non-saturated, nodes as long as the number of faulty components or bottlenecks is less than d, the degree of the hypercube.

In Algorithm GHC-P, the pathlist is sent along with the message packet to indicate the destination of a packet. In addition to the pathlist, a set containing those nodes on the first coordinate of the pathlist, which have already been visited is also sent with the packet. This set which is called the *visited_nodelist* will clear (becomes $\emptyset$) whenever the packet is routed into a new dimension. Additionally, each packet is accompanied with an r-element set *tag*. The i-th element of tag corresponds to the i-th coordinate of the GHC and is an m_ith-tuple which has one digit corresponding to every node of the i-th coordinate. The tag keeps track of "spare dimensions and links" that are used to bypass faulty or saturated components. All bits in the tag are reset to zero when the source node begins the routing of a packet. In our notations, *tag(c, n)* is the n-th bit of the c-th coordinate, (c-th element) of tag. A packet can be represented as *(k, pathlist, visited_nodelist, message, tag)*, where k is the length of the remaining portion of the path and the entire thing is updated as the message travels towards the destination. A packet reaches its destination when $k = 0$, or pathlist becomes $\emptyset$.

When a node receives a packet, it will check k to see if the node is the destination of the packet. If not, the node will try to send the packet along one of those links specified in the remaining elements of *pathlist*. The k and pathlist are updated as the packet travels though the hypercube. Each node will initially attempt to route messages via shortest paths. If the link in the dimension specified by a pair in *pathlist*

```
/*  At each node $(k, [(c_1, n_1), (c_2, n_2), \ldots, (c_k, n_k)], visited\_nodelist, message, tag)$ */
/*  In this algorithm, $\odot$ denote an append operation                                        */

if $k = 0$ then {the destination is reached!}
else
begin
   /*  Try to send the packet along a dimension in
       the remaining coordinate sequence.          */
   for $j := 1, k$ do
       if (the $(c_j, n_j)$ link is not faulty) then
          send $(k - 1, [(c_1, n_1), \ldots, (c_{j-1}, n_{j-1}), (c_{j+1}, n_{j+1}), \ldots, (c_k, n_k)],$
               $\emptyset, message, tag)$ along the $(c_j, n_j)$ link;
          stop; /* Terminate Algorithm GHC-P */
       else if $\exists (c_j, y) \mid (c_j, y)$ is not faulty and $y \notin visited\_nodelist$ then
          send $(k, [(c_1, n_1), (c_2, n_2), \ldots, (c_k, n_k)], visited\_nodelist \odot x_{c_j},$
               $message, tag)$ along the $(c_j, y)$ link
       /* NOTE: $x_{c_j}$ is the $c_j$th digit of the address of the current node */
          stop; /* Terminate Algorithm GHC-P */
       end_if
   end_do

   /*  If the algorithm is not terminated yet, all dimensions in
       the pathlist are blocked because of faulty components
       and a spare dimension needs to be used.*/

   for  $j$ := 1, $k$ do        /* Record all blocked
        $tag(c_j, x_{c_j}) := 1$;          links in tag.  */
   end_do
   $h := c \& y := n \mid (tag(c, n) = 0 \& n = min, 1 \le c \le d, 1 \le n \le m_c \& n \ne x_h)$
              /* Choose a spare dimension */

   $tag(h, y) := 1$; /* update the tag */
   send $(k + 1, [(c_1, n_1), (c_2, n_2), \ldots, (c_k, n_k), (h, x_h)], \emptyset, message, tag)$
                              along the $(h, y)$ link
   stop;        /* Terminate Algorithm GHC-P */
end_begin
```

Figure B.2: Algorithm GHC-P – Adaptive routing algorithm to be used by each node of a GHC only with the information on its own links.

is faulty, the algorithm attempts to send the packet through another link in the same dimension and appends the name of the current node to the *visited_nodelist*. In this case, k and pathlist stay the same and are not modified. When the packet is sent through a new coordinate, *visited_nodelist* is cleared. However, if all the links in those dimensions on the *pathlist* are faulty, the node will use a spare dimension to route the packet via an alternate path. The *tag* keeps track of the available spare dimensions and links. Listing B.2 is a more formal presentation of Algorithm GHC-P.

In the GHC in Figure B.1, the links drawn with dashed lines are faulty or congested. Suppose a message is routed from A = 021 to B = 300. The packet at every node on the path will be:

@ 021 ← (3, [(1,0),(2,0),(3,3)], Ø, message, [0000,000,00])
@ 020 ← (2, [(2,0),(3,3)], Ø, message, [0000,000,00])
@ 010 ← (2, [(2,0),(3,3)], [2], message, [0000,000,00])
@ 310 ← (1, [(2,0)], Ø, message, [0000,000,00])
@ 320 ← (1, [(2,0)], [1], message, [0000,000,00])
@ 321 ← (2, [(2,0),(1,0)], Ø, message, [0000,001,10])
@ 311 ← (2, [(2,0),(1,0)], [2], message, [0000,001,10])
@ 301 ← (1, [(1,0)], Ø, message, [0000,001,10])
@ 300 ← (0, Ø, Ø, message, [0000,001,10])

Flow Control: Due to its progressive nature, GHC-P algorithm can take advantage of the low latency of the wormhole routing. However, since we have relaxed the dimension-order routing and have allowed messages to flow from one dimension to another in either direction (in contrast with deterministic routings,) we have created a channel dependency graph that may result in deadlock. For example, in Figure

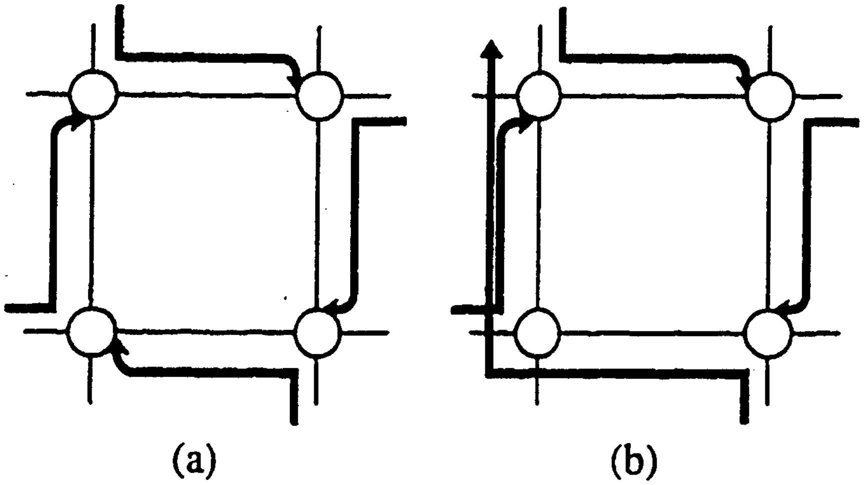

Figure B.3: (a) Deadlock: (b) Breaking the deadlock using a north-bound virtual channel for east-bound packets moving north-bound.

B.3(a) each of the four packets require the channel occupied by the channel ahead in the cycle. and the network is therefore deadlocked.

The cyclic dependency can be broken by restricting routing so that east-bound packets (positive x-direction) are not allowed to travel on north-bound (positive y-direction) channels. With this restriction. deadlock is no longer possible. but the network has become disconnected. The connectivity can be restored by introducing a north-bound virtual channel for east-bound messages [Figure B.3(b)]. The resulting network performs deadlock-free adaptive routing.

To scale this mechanism up to the general case. as each dimension. i. is added to the network, traffic in the previous dimension is divided into 2^{i-1} groups according to its directions in the $i-1$ previous dimensions. One direction of travel in the i-th dimension is then partitioned into 2^{i-1} virtual channels, one for each group.

B.2 Adaptive Routing in *WK*-Recursive Networks

The recursive structure and low number of links per node in WK-recursive networks make them ideal candidates for massively parallel computers. Due to the large number of processors in these machines. adaptive routing algorithms which implement backtracking or delay-tables are absolutely impractical. Therefore. there is a big demand for a progressive adaptive routing algorithm which can route messages in these networks efficiently and reliably.

Algorithm WKR This algorithm will route messages in a *WK*-Recursive network in the presence of an arbitrary number of failures or bottlenecks. as long as there is a path from the source to the destination. Each node is only required to be aware of the condition (faulty congested) of its own links. A messages in this routing is represented as $(d, td, Visited\_Nodes, message)$ in which d is the destination address. td is a tag word $L-1$ digits long and each digit corresponds to one of the $2:L$ dimensions of the structure. td stores the temporary destinations that the message has to go through to bypass a failure or congestion in the network. $Visited\_Nodes$ is a $(L-1)\times k$ array which stores the addresses of the visited nodes inside a virtual node. After the message leaves the virtual node. the components in the array corresponding to the nodes which are of lower dimension are all cleared. Listing 6.5 is the complete WKR algorithm.

Figure B.4 shows a *WK*-Recursive network with $L = 3$ and $k = 4$. The links drawn with dashed lines are faulty or congested. Suppose a message is routed from $S = 002$ to $D = 202$. The transferred packet at every node on the path will be:

NODE ←		PACKET SENT	LINK NUMBER
@ 002	←	(202, XX, [0, 0], message)	2
@ 020	←	(202, XX, [0, 02], message)	2
@ 022	←	(202, 1X, [0, 2], message)	1
@ 021	←	(202, 10, [0, 2], message)	2

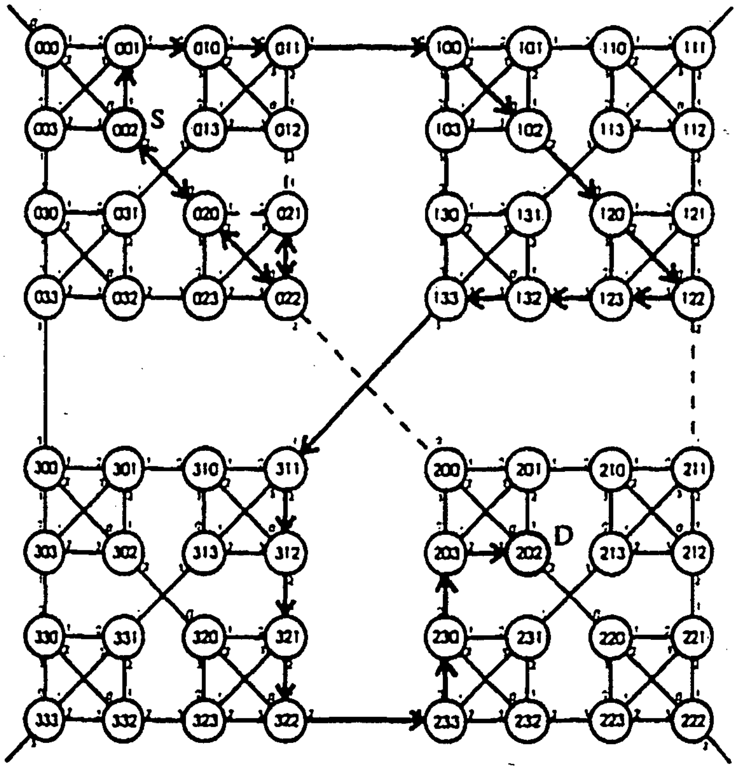

Figure B.4: Adaptive routing in a *WK*-Recursive network with $k = 4$ and $L = 4$ and four faulty or congested links.

@ 022	←	(202, 10, [0, 2], message)	0
@ 020	←	(202, 10, [0, 2], message)	0
@ 002	←	(202, 1X, [0, 02], message)	1
@ 001	←	(202, 1X, [0, 02], message)	1
@ 010	←	(202, 1X, [0, 012], message)	1
@ 011	←	(202, 1X, [0, 012], message)	1
@ 100	←	(202, XX, [01, 0], message)	2
@ 102	←	(202, XX, [01, 0], message)	2
@ 120	←	(202, XX, [01, 02], message)	2
@ 122	←	(202, 3X, [01, 2], message)	3
@ 123	←	(202, 3X, [01, 2], message)	3
@ 132	←	(202, 3X, [01, 23], message)	3
@ 133	←	(202, 3X, [01, 23], message)	3
@ 311	←	(202, XX, [013, 1], message)	2
@ 312	←	(202, XX, [013, 1], message)	2
@ 321	←	(202, XX, [013, 12], message)	2
@ 322	←	(202, XX, [013, 12], message)	2
@ 233	←	(202, XX, [0123, 3], message)	0
@ 230	←	(202, XX, [0123, 3], message)	0
@ 203	←	(202, XX, [0123, 03], message)	2
@ 202	←	REACHED DESTINATION	

```
/*  Compare&Get_Link($dest$) procedure returns the link number which is equal to
    the most significant digit of $dest$ which is different from node address
    $CNA$ = Address of the current node                                          */

receive ($d$,$td$,$Visited\_Nodes$,message)
if $d = CNA$ then {the destination is reached!}
else
begin
  for $i = 1 : L - 1$
      if $td(i) \neq X$ then
        if $td(i) = CNA(i)$ then $td(i) := X$
        break
      end_if
  end_for
  if $td = \emptyset$then
      $Link$ = Compare&Get_Link($d$)
  else
      $Link := td(i)$
  end_if
  if ($Link$ is faulty) then
      for $level = 1 : L$
        if $CNA(level) \neq Link$ then break
      end_for
      for $Link = 0 : k - 1$
        if $Link \notin Visited\_Nodes(level)$ then break
      end_for
      $td(level) := Link$
      for $i = 1 : level$
        clear($Visited\_Nodes(i)$)
      end_for
  end_if
  for $i = 2 : L$
      $Visited\_Nodes(i) := CNA(i) \odot Visited\_Nodes(i)$
  end_for
  send ($d$,$td$,$Visited\_Nodes$, message) along $Link$
end_begin
```

Figure B.5: Algorithm WKR - Adaptive routing algorithm to be used by each node only with the information on its own links.

REFERENCES

[1] A. Agarwal. Limits on interconnection network performance. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):398–412, October 1991.

[2] G. Alverson, R. Alverson, D. Callahan, B. Koblenz, A. Porterfiled, and B. Smith. Exploiting heterogeneous parallelism on a multithreaded multiprocessor. In *Proceedings of the 6th ACM International Conference on Supercomputing*, 1992.

[3] Ahmad R. Ansari and Fred J. Taylor. UF^3 - a 4D DSP hypercube with a robust programming environmment. In *Proceedings of International Conference on Acoustics, Speech, and Signal Processing*, volume V, pages 633–636, San Francisco, California, March 1992.

[4] C. M. Aras, J. F. Kurose, D. S. Reeves, and H. Schulzrinne. Real-time communication in packet-switched networks. *Proceedings of the IEEE*, 82(1):122–139, January 1994.

[5] J. R. Armstrong and F. G. Gray. Fault-diagnosis in a boolean n-cube array of microprocessors. *IEEE Transactions on Computers*, C-30(8):587–590, August 1981.

[6] W. C. Athas and C. L. Seitz. Multicomputers: Message-passing concurrent computers. *IEEE Computer*, 21(8):9–24, August 1988.

[7] J. J. Bae and T. Suda. Survey of traffic control schemes and protocols in ATM networks. *Proceedings of the IEEE*, 79(2):170–189, February 1991.

[8] R. Boppana and S. Chalasani. A comparison of adaptive wormhole routing algorithms. In *Proceedings of International Symposium on Computer Architecture*, pages 351–360, 1993.

[9] R. M. Bryant, H. Y. Chang, and B. S. Rosenburg. Operating system support for parallel programming on RP3. *IBM J. Res. Develop.*, 35(5/6):617–634, Sep/Nov 1991.

[10] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina. Architectural requirements of parallel scientific applications with explicit communication. In *Proceedings of the International Symposium on Computer Architecture*, pages 2–13, May 1993.

[11] W. J. Dally. *A VLSI Architecture for Concurrent Data Structures.* Kluwer Academic Publishers, Boston, MA, 1987.

[12] W. J. Dally. Network and processor architecture for message-driven computers. In Birtwistle Suaya, editor, *VLSI and Parallel Computation*, chapter 3. Morgan Kaufmann, San Mateo, California, 1990.

[13] W. J. Dally. Performance analysis of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 39(6):775–785, June 1990.

[14] W. J. Dally. Express cubes: Improving the performance of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 40(9):1016–1023, September 1991.

[15] W. J. Dally. Virtual channel flow control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, March 1992.

[16] W. J. Dally and H. Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466–475, April 1993.

[17] W. J. Dally, S. Fiske, J. S. Keen, R. A. Lethin, M. D. Noakes, P. R. Nuth, R. E. Davidson, and G. A. Fyler. The message-driven processor: A multicomputer processing node with efficient mechanism. *IEEE Micro*, pages 23–29, April 1992.

[18] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[19] Michael L. Dertouzos and Aloysius Ka-Lau Mok. Multiprocessor on-line scheduling of hard-real-time tasks. *IEEE Transactions on Software Engineering*, 15(12):1497–1506, December 1989.

[20] J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, December 1993.

[21] Ronald Fernandes. Recursive interconnection networks for multicomputer networks. In *Proceedings of the 1992 International Conference on Parallel Processing*, volume I, pages 76–79, 1992.

[22] Domenico Ferrari. Client requirements for real-time communication services. *IEEE Communication Magazine*, pages 65–72, November 1990.

[23] D. Gelernter. A DAG-based algorithm for prevention of store-and-forward deadlock in packet networks. *IEEE Transactions on Computers*, C-30(10):709–715, October 1981.

[24] C. J. Glass and L. M. Ni. The turn model for adaptive routing. In *Proc. 19th Int'l Symp. Computer Architecture, Los Alamitos, CA*, pages 278–287, New York, 1992. IEEE CS Press.

[25] A. Gottlieb, R. Grishman, C. P. Kruskal, K. P. McAuliffe, L. Rudolf, and M. Snir. The NYU ultracomputer - designing an MIMD shared memory parallel computer. *IEEE Transactions on Computers*, C-32(2):175–189, February 1983.

[26] K. D. Gunther. Prevention of deadlocks in packet-switched data transport systems. *IEEE Communication Magazine*, COM-29(4):512-524, April 1981.

[27] M. T. Heath. The hypercube: A tutorial overview. In *Proceedings of the Second Conference on Hypercube Multiprocessors, Knoxville, TN*, pages 7–10, 1986.

[28] V. Karamcheti and A. A. Chien. Do faster routers imply faster communication? In *Proc. Parallel Computer Routing and Communication Workshop, Seattle, WA*, pages 1–15, Berlin, 1994. Springer-Verlag.

[29] H. Katseff. Incomplete hypercubes. *IEEE Transactions on Computers*, C-37(5):604–608, May 1988.

[30] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks*, 3(4):267–286, September 1979.

[31] Leonard Kleinrock. *Queueing Systems*, volume 1. Wiley-Interscience, New York, 1975.

[32] Clyde P. Kruskal and Marc Snir. The performance of multistage interconnection networks for multiprocessors. *IEEE Transactions on Computers*, C-32(12):1091–1098, December 1983.

[33] D. Lenoski, K. Gharachorloo, J. Laudon, A. Gupta, J. Henessy, M. Horowitz, and M. Lam. Design of scalable shared-memory multiprocessors: The dash approach. In *Proceedings of COMPCON*, pages 62–67, 1990.

[34] Clement H. C. Leung. *Quantitative Analysis of Computer Systems*. John Wiley and Sons, New York, 1988.

[35] C. L. Liu and James W. Layland. Scheduling algorithms for multiporgramming in a hard real-time environment. *Journal of ACM*, 20(1):46–61, January 1973.

[36] Michael K. Molloy. *Fundamentals of Performance Modeling*. Macmillan, New York, 1989.

[37] NCUBE, 919 East Hilldale Boulevard, Foster City, CA. *nCUBE 2 Programmer's Guide*, 1992.

[38] J. Y. Ngai. *A Framework for Adaptive Routing in Multicomputer Networks*. PhD thesis, California Institute of Technology, Pasadena, 1989. Caltech-CS-TR-89-09.

[39] G. F. Pfister. The IBM research parallel processor prototype (RP3): Introduction and architecture. In *Proceedings of the ICPP*, pages 764–771, August 1985.

[40] D. A. Reed and R. M. Fujimoto. *Multicomputer Networks: Message-Based Parallel Processing*. MIT Press, Cambridge, MA, 1987.

[41] D. A. Reed and H. D. Schwetman. Cost-performance bounds for multimicrocomputer networks. *IEEE Transactions on Computers*, C-32(1):83-95, January 1983.

[42] C. L. Seitz. The cosmic cube. *Communications of ACM*, 28(1):22-33, January 1985.

[43] Kang G. Shin and Parameswaran Ramanathan. Real-time computing: A new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1):6–24, January 1994.

[44] S. Toueg. Deadlock- and livelock-free packet switching networks. In *Proc. 12th ACM Symp. Theory of Computing*, pages 94–99, 1980.

[45] S. Toueg and J. D. Ullman. Deadlock-free packet switching networks. In *Proc. 11th ACM Symp. Theory of Computing*, pages 89–98, 1979.

[46] G. Della Vecchia and C. Sanges. Recursively scalable networks for message passing architectures. In *Parallel Processing and Applications*, pages 33–40, Amsterdam, September 1987. Elsevier Science Publishers B.V.

[47] G. V. Wilson. A glossary of parallel computing terminology. *IEEE Parallel and Distributed Technology Magazine*, 1(1):52-67, February 1993.

[48] L. D. Wittie. Communications structures for large networks of microcomputers. *IEEE Transactions on Computers*, C-30(4):264–273, April 1981.

[49] X. Zhang. System effects of interprocessor communication latency in multicomputers. *IEEE Micro*, pages 12–15, 52–55, April 1991.

[50] Wenjing Zhu and Sameul T. Chanson. Adaptive threshold-based scheduling for real-time and non-real-time traffic. In *Proceedings of the 12th Real Time System Symp.*, pages 125–135, 1992.